ISSN 0998-9250 Mai 1991 / N° 12

Avec cahier Dr Dobb's Journal

DEBUG magazine

MAITRISEZ VOTRE DOS

GESTION MEMOIRE
CACHE DISQUE
PILOTES EN C

Analyse fonctionnelle
Algorithmique

EVALUATION
Paradox Engine
High Screen 5

EXCLUSIF
Turbo Pascal Windows

M3032 - 12 - 37,00 F

Suisse : 11,5 FS
Belgique : 258 FB

3614 DEBUG
LANCEMENT LE 15 MAI 1991

E SERVICE TECHNIQUE POUR LES PROFESSIONNELS

514 DEBUG est un nouveau service télématique, spécialement crée pour vous.
uisez dans les ressources de DEBUG MAGAZINE, immédiatement et facilement.

E LIEU DE RENCONTRE DES EXPERTS

514 DEBUG est l'endroit de rencontre d' experts, comme vous. La messagerie permet de discuter, lancer des forums, échanger des idées.
vous ne trouvez pas de réponse ni dans les livres ni dans les manuels, posez la question à d'autres : la réponse peut arriver instantanément!

OMMUNIQUEZ AVEC LA RÉDACTION

514 DEBUG est un moyen facile pour dialoguer avec la rédaction. Posez des questions, suggérez des idées, postez vos lettres, commentaires et pinions.

OTRE ESPACE PUBLICITAIRE GRATUITE

ue vous ayez un lecteur à échanger contre une carte ou que vous cherchiez une pièce détachée rare, un équipement d'occasion à prix réduit, pro-ez des petites annonces **3614 DEBUG.** Un partagiciel à annoncer, un logiciel à proposer en vente, vous cherchez un éditeur, programmeur, nsultant ou dépanneur - profitez de leur présence sur **3614 DEBUG,** annoncez vos offres et vos besoins. Vos petites annonces seront lues instan-nément, sans attendre la sortie mensuelle du magazine. **3614 DEBUG,** c'est instantané et ça peut rapporter gros!

ELECHARGEZ

514 DEBUG est un moyen rapide pour télécharger les listings de DEBUG MAGAZINE. Vous pouvez télécharger tous les sources et exécutables vec le logiciel OVERCOM (émulation Minitel, TTY, VT52/VT100, protocoles GOTO, Kermit, X et Y modem, SM1, langage de procédure, auto-pprentissage. Fonctionne sur toutes les cartes modem et avec un Minitel, livré avec câble Minitel, adaptateur DB9/DB25, manuel 300 pages). ontactez-nous : prix spécial pour les lecteurs de DEBUG MAGAZINE!

ENDANT LE LANCEMENT LE SERVICE EST GRATUIT!

endant deux mois de démarrage, vous pouvez utiliser ce service gratuitement - seul le coût de communication sera à votre charge (tarif 3614).

EDITO

LE BORGNE ET L'AVEUGLE

Maîtrisez votre Dos ! Un voeux dirigé au grand maître de DOS. Depuis quelques temps Microsoft a du mal à maîtriser son cheval de bataille préféré. Les systèmes opérationnels, applicatifs, interfaces graphiques, bases de données, réseaux locaux, les projets pullulent, les bons produits se font rares.

Prenons, justement, MS - DOS. La version 5, annoncée en catastrophe pour contrer la concurrence, n'est toujours pas disponible. Le seul DOS qui existe réellement est celui de Digital Research. Il est disponible, marche bien et exploite intelligemment les machines modernes. Celui de Microsoft, circulant sous forme bêta (donc en diffusion gratuite), n'est toujours pas prêt.

Vous allez dire qu'un DOS n'est pas une révolution, c'est Windows qui est plus important. Regardons de plus près. Ce qui devrait être une mémoire, les périphériques et Dieu sait quoi encore. C'est comme si un aveugle portait le borgne. MS - DOS ne peut pas, trop vieux et pas adapté, voir les machines modernes. Windows, handicapé par l'absence de gestion de disques, ne sait pas courir tout seul. L'un est assis sur le dos de l'autre, ils marchent ensemble. « Et toi, fais moi ouvrir un fichier », « quoi, un fichier », une tâche passe, le fichier n'est toujours pas ouvert; « oui, un fichier, ouvre vite », tout se bloque, boum !

Pendant ce temps, les sages de Redmont comptent le nombre de diables sur la pointe d'un stylo et s'interrogent comment trafiquer OS/2 pour concurrencer Go Pen.

Et si on s'occupait un peu d'un bon DOS ? Si l'on créait un seul et unique système multi-tâche (dans le style de Multiuser DOS de Digital Research), dans chaque tâche une interface compatible Windows, compatible Novell, bref un système qui marche ?

Mais, c'est vrai, les anges passent, les diables restent. Et qui a bon DOS, dort bien...

* ART *

PARLONS - EN

☛ En réponse à l'article paru dans le n° 11, la société AMSTRAD nous fait parvenir le texte suivant :

Un prospectus est un document informatif, non contractuel, permettant d'orienter le choix du consommateur. Si il fait l'objet d'une coquille ou d'une erreur d'impression, ce qui est le cas dans l'affaire de Monsieur ROUICHI de Garges les Gonnesses, un rectificatif dans la presse, et dans le réseau de revendeurs AMSTRAD par courrier et par information directe via les ingénieurs commerciaux d'AMSTRAD est fait systèmatiquement.

Il fallait lire « emplacement pour co-processeur » et non « co-processeur », le mot « emplacement » ayant sauté de la maquette du prospectus au moment de la photogravure. Cette coquille n'a d'ailleurs figuré que sur une centaine de prospectus diffusés lors du premier tirage.

Monsieur ROUICHI attend d'AMSTRAD des excuses, se répand dans la presse en lettres et propos désobligeants concernant la marque AMSTRAD.

Monsieur ROUICHI téléphone content du résultat de son action aux collaborateurs de la Société. Monsieur ROUICHI est devenu célèbre, pense-t-il. Alors que son incompétence en tant qu'ex vendeur de micro-informatique a fait qu'il a acheté en tant que responsable d'association, un PC AMSTRAD n'ayant pas et pour cause, un coprocesseur arithmétique dont est seulement fourni en standard l'emplacement.

Monsieur ROUICHI fait de l'esprit, critique le service Marketing d'AMSTRAD, cite des noms de collaborateurs qui ont fait pour lui être agréable malgré ses menaces répétées et personnelles, sans se substituer cependant au revendeur AMSTRAD, qui lui, a rectifié sans aucun problème la coquille, proposant même sur notre conseil de lui reprendre l'appareil si la présence d'un co-processeur arithmétique était un critère déterminant pour lui dans son choix.

Le revendeur d'AMSTRAD lui, se tient au courant, lit la presse AMSTRAD ou autre, les bancs d'essai, les comparatifs, les nomenclatures du marché, les documentations techniques, les manuels d'utilisation, etc... Lorsqu'il vend un nouveau modèle, il ne se contente pas de lire un prospectus, il découvre la machine, il l'ouvre et la teste pour être à même de répondre à son client et de le conseiller au mieux... Il a une vraie mission de conseil et sait ce qu'il faut rectifier dans le cas d'une erreur de son fournisseur.

Monsieur ROUICHI sait parfaitement cela, en tant qu'ex vendeur informatique : il ne lui reste donc qu'à trouver des rubriques lecteurs au demeurant fort intéressantes pour avouer en définitive son incompétence et sa fausse bonne foi, avec le soutien très impartial de SVM et très partial de DEBUG : le premier a eu la courtoisie et l'objectivité propre aux journalistes de vérifier et d'étayer l'information auprès des personnes nommément citées voire diffamées, ce qui ne fut pas le cas du commentateur anonyme du second.

Monsieur ROUICHI aura par son indignation bien légitime et son obstination à entretenir un faux procès d'intention, contribué à rectifier plus largement encore dans la presse la coquille d'AMSTRAD, c'est un fait ; coquille rectifiée d'office par l'ensemble du réseau de revendeurs d'AMSTRAD sans aucune contestation et de leur part et de la part des 3 000 acquéreurs de PC 3286 depuis leur sortie.

Monsieur ROUICHI aura démontré que ce n'est pas parce que l'on se dit spécialiste de la micro-informatique que l'on doit se dispenser du conseil éclairé d'un revendeur ou d'un article de presse, tel qu'un excellent banc d'essai paru sur ce matériel avant de l'acquérir. Ce qui justifie la vocation d'informer de la presse informatique, et le soutien qu'AMSTRAD lui apporte.

AMSTRAD a tout au long de cette affaire répondu avec tact et courtoisie aux questions de Monsieur ROUICHI et ne saurait accepter de voir mettre en cause la qualité de son équipe marketing et de ses collaborateurs.
Geneviève Metzdorf
Directrice Marketing
Amstrad France.

NDLR : Sur la demande expresse de la société AMSTRAD, nous publions ici cette surprenante lettre. Quand à nous, nous pensons que le fond du problème n'est pas la personne de Mr Rouichi ni ses qualifications professionnelles (dont personne d'ailleurs ne peut se porter juge) mais l'attitude de la société AMSTRAD face à un client et consommateur. Les lecteurs vont apprécier.

☛ ***« (...) dans l'interview de Mr Gates du n° 11, (...) on se demande si des lignes n'ont pas disparu comme page 16, 3ème colonne dans le paragraphe où apparaît Mr Kahn avec la question fondamentale : « Qui a commencé Adobe ou Borland? » Ce qui serait très intéressant, serait de préciser qui est ce J Warnock, pour ma part je pense que c'est le gérant de la station Mobil qui est en face du siège social de Borland, mais qu'il s'est installé après lui dans la rue. Mais je vous encourage, il faut continuer ! »***
Régis Pidoux,
Sté MER à la Trinité sur Mer

[SERVICE LECTEURS : INTERESSANT = 1 / MOYEN = 2/ MAUVAIS = 3]

ABONNEMENT 1 AN DISQUETTES LISTING

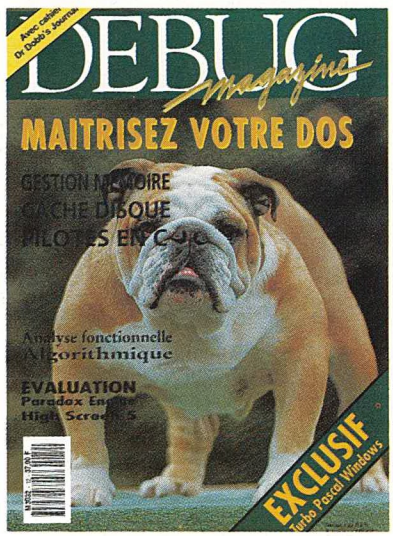
❒ Je m'abonne pour **12** disquettes listing : **399 F** au lieu de **444 F**
Je règle par : ❒ chèque à l'ordre de **Debug Magazine**
❒ carte bancaire
Nom de la carte : .....
N° ..../..../..../..../..../..../..../..../..../..../..../..../..../..../ Date d'expiration /..../..../..../..../
Nom :..... Prénom : .....
Société :.....
Rue :.....
Lieu :.....
Code postal :..... Ville : .....
Pays :..... Tél : .....
Date :..... Signature : .....

A renvoyer à **DEBUG MAGAZINE** - 3, rue de l'Atlas - 75019 Paris

COMMANDE DISQUETTE LISTING N° 12

❒ Je désire recevoir la disquette listing **DEBUG 12** au prix de **47 F** pièce

Je règle par : ❒ chèque à l'ordre de **Debug Magazine**
❒ contre remboursement (ajouter **40 F** de frais)
❒ carte bancaire
Nom de la carte : .....
N° ..../..../..../..../..../..../..../..../..../..../..../..../..../..../ Date d'expiration /..../..../..../..../
Nom :..... Prénom : .....
Société :.....
Rue :.....
Lieu :.....
Code postal :..... Ville : .....
Pays :..... Tél : .....
Date :..... Signature : .....



Anciennes disquettes à 47 F: ❒ n° 1 ❒ n° 2 ❒ n° 3 ❒ n° 4 ❒ n° 5 ❒ n° 6 ❒ n° 7 ❒ n° 8
❒ n° 9 ❒ n° 10 ❒ n° 11 ❒ Hors - Série n° 1
❒ Disquettes utilitaires : PKPAK - LHARC - LZE, LZESHELL

A renvoyer à **DEBUG MAGAZINE** - 3, rue de l'Atlas - 75019 Paris

SERVICE LECTEURS

DEBUG Magazine n° 12

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Nom : ..... Prénom : .....
Société : .....
Rue : .....
Lieu : .....
Code postal : ..... Ville : .....
Pays : ..... Téléphone : .....

A REMPLIR OBLIGATOIREMENT SI VOUS AVEZ DEMANDÉ UNE DOC.

Le Service Lecteurs de DEBUG Magazine transmettra aux intéressés les demandes d'envoi de documentation dont vous aurez coché les cases.

Notre service vous permet également de donner votre opinion sur les articles parus. Cochez les cases correspondant à votre jugement sur l'article concerné (les petits numéros au début de chaque article, pas les numéros des pages) ; cela nous permettra d'améliorer la qualité de la revue. Merci !

pliez ici en second et collez l'autre côté au ruban adhésif.

AFFRANCHIR
AU TARIF
LETTRE..

DEBUG MAGAZINE
3, rue de l'Atlas
75019 Paris

pliez ici en premier.

QUESTIONNAIRE – LECTEUR

Année de naissance : 19__
Sexe : ☐ M ☐ F

Votre secteur d'activité :

- ☐ Agriculture
- ☐ Enseignement
- ☐ Autres Fonction Publique, Collectivité locale
- ☐ Association
- ☐ Energie
- ☐ Métallurgie
- ☐ Chimie, verre, plastique
- ☐ Construction automobile, mécanique, navale, aéronautique
- ☐ Electronique
- ☐ Bâtiment, travaux publics
- ☐ Autres secteurs industriels
- ☐ Commerce
- ☐ Transports
- ☐ Télécommunications
- ☐ SSII
- ☐ Autres services aux entreprises
- ☐ Hôtellerie, réparation et autres services aux particuliers
- ☐ Banque, Assurance
- ☐ Autres secteurs commerciaux

Nombre de salariés dans l'établissement : ____

Votre profession :

- ☐ Propriétaire
- ☐ PDG-DG-Gérant
- ☐ Cadre
- ☐ Employé
- ☐ Ouvrier
- ☐ Retraité
- ☐ Autre

Votre matériel :

- ☐ IBM ou compatible
- ☐ Apple, MAC
- ☐ Mini
- ☐ Mainframe

L'utilisation de votre matériel :

- ☐ privée
- ☐ professionnelle
- ☐ les deux

Etes-vous satisfait de votre équipement actuel ? ☐ Oui ☐ Non
Allez-vous l'étendre ? ☐ Oui ☐ Non

Vos commentaires :

"Qui selon vous pouvait vous offrir le meilleur compilateur pour la création d'applications Windows ?"

– Réponse ci-dessous

Maintenant il y a un moyen nettement supérieur pour écrire des applications Windows. Le nouveau Borland® C++ est le seul environnement complet de programmation C et C++ pour DOS et Windows. Borland C++ génère des applications Windows et des DLL(s), gère le débogage sous Windows et inclut un éditeur de ressources visuel. Vous n'avez donc pas besoin du SDK (le Kit de Développement de Microsoft Windows).

Borland C++ a été conçu par des spécialistes qui connaissent les besoins des programmeurs professionnels. En fait Turbo C++ Professionnel, le prédécesseur de Borland C++, a remporté à la fois le prix d'Excellence Technique de PC Magazine et le prix d'Excellence de Byte en 1990.

Conçu pour des professionnels par des professionnels

Chaque exemplaire de Borland C++ est fourni avec :

- Un compilateur ANSI C et un compilateur C++.
- Un compilateur en ligne de commande Turbo Drive et un environnement de développement intégré fonctionnant en mode protégé pour une plus grande capacité.
- Des fichiers à en-tête précompilés qui augmentent considérablement la vitesse de recompilation.
- Turbo Debugger pour DOS et Windows.
- Whitewater Resource Toolkit pour la création d'icônes, de boîtes de dialogues, de graphiques en mode point et de menus.
- Turbo Profiler et Turbo Assembler.

A un prix compétitif

	Borland C++
Compilateur C	4995 F HT
Compilateur C++	inclus
Outil de programmation Windows	inclus
Assembler	inclus
Total	**4995 F HT**

Commandez-le dès maintenant Nous vous offrons *"Programmer sous Windows 3"*,

Achetez tout de suite Borland C++ chez votre revendeur habituel et vous bénéficierez gratuitement* d'un exemplaire de *"Programmer sous Windows 3"* de Charles Petzold, le best-seller de l'apprentissage de la programmation Windows.

Des conditions spéciales de mise à jour sont faites par Borland, pour les détenteurs de Turbo C, Turbo C Professionnel, Turbo C++ ou Turbo C++ Professionnel.

Rendez vous chez votre revendeur habituel dès aujourd'hui ou téléphonez chez Borland pour une mise à jour au

N° VERT 05.46.96.69

APPEL GRATUIT

**Dans la limite des stocks disponibles, offre valable jusqu'au 15 juin 1991.*

La réponse :

BORLAND

Le leader de la Programmation Orientée Objets

43, avenue de l'Europe – BP 6 – 78143 Vélizy-Villacoublay – France – Tél. (33) (1) 39.46.96.69 – Télécopie (33) (1) 39.46.81.60 – Télex 698 793

La question de Bill, telle qu'elle est imprimée, est : "Qui a commencé LE PREMIER, Adobe ou Borland?", ce qui se passe, nous pensons, de tout explication. Quand à John Warnock, à côté de Gates, Kahn, Kapor, Allen, Barlow ou Jobs (pour ne nommer que quelques « loups blancs » de l'industrie), il est le fondateur de la société Adobe Systems, crée il y a une dizaine d'années. John a inventé ce qui est peut-être la plus importante innovation dans le domaine de l'impression, depuis que Gutenberg a inventé l'impression avec des lettres mobiles : le système automatique de génération de fontes sur les ordinateurs personnels. Dans sa démarche, il a été suivi par Paul Brainerd qui s'est assis dans sa cuisine et a crâné sur le concept de pilotage de toute une imprimerie à partir d'un PC. Tout cela a donné la naissance à Pagemaker (Aldus Corp.), suivi par une multitude d'autres logiciels de PAO, qui ont totalement transformé la profession de typographe et d'imprimeur... C'est aussi grâce à ces deux personnages que vous pouvez lire aujourd'hui ces lignes, intégralement crées et composées en utilisant les technologies nouvelles. Sans PAO, DEBUG MAGAZINE simplement n'existerait pas - nous (et vous!) pouvons l'affirmer. En effet, une (seule...) fois nous avons tenté de réaliser le magazine avec des techniques traditionnelles - cela a donné le n° 7, dont on se rappelle encore la qualité typographique (ou plutôt son absence) et le délai de réalisation...

DEBUG MAGAZINE publie les interviews avec les certaines personnalités, pour permettre aux lecteurs intéressés de découvrir l'histoire de l'industrie informatique et ses principaux acteurs.

☛ ***« J'ai appris dernièrement que l'on pouvait augmenter la capacité des disquettes 3"1/2 DSDD aux disquettes HD. Fort de cette nouvelle astuce, je décidais de sacrifier une de mes disquettes DD pour la soumettre à des tests. J'ai donc retiré le petit cache inférieur droit (là ou se trouve le trou sur les disquettes HD) avec mon couteau suisse. Puis je l'ai formaté (sous MS-DOS 5.0 Bêta, 4.0 et 3.3) sans aucun problème, ce qui m'a rendu 1.44 Mb de capacité. Restait à savoir si elle allait supporter cette nouvelle puissance (que la force soit avec toi!). J'ai donc édité un petit programme qui écrit et vérifie des fichiers que je lui met à disposition (.exe, .com, .doc, .gif etc.).***

Après 1 heure d'intense transfert, le programme n'a pas détecté une seule erreur.

Viens maintenant la grande question, à savoir, est-ce que les fabricants des disquettes sont vraiment aussi idiots de fabriquer une seule disquette et de plomber celles qui auront l'étiquette DD. Ainsi, nous serions stupides à notre tour de payer plus cher celles qui se trouvent dans les boites HD. Je pense que vous serez plus apte à répondre. Merci d'avance et j'attends impatiemment le prochain numéro ».

Yves Baggi,
Onex (Geveve - Suisse)

Commençons par le commencement. Sur une disquette 360 Ko les données sont enregistrées sur 40 pistes, chaque piste divisée en 9 secteurs de 512 bits chacun - c'est cela qui s'appelle « disquette standard PC », crée par IBM pour son modèle PC. Mais il est bien connu, qu'IBM est la seule entreprise qui crée des standards et qui, ensuite, ne le respecte pas. C'est ainsi que lors de lancement de modèle AT, IBM a crée un second standard : disquette « AT », divisée en 80 pistes, divisées en 15 secteurs. Sur la même surface, on a doublé le nombre de pistes et augmenté le nombre de secteurs. Evidemment, les disquettes « AT » ne peuvent pas être lues sur les lecteurs « PC » bien que l'inverse est possible. L'écriture (enregistrement) sur les disquettes au format « PC » dans un lecteur « AT » n'est pas ni fiable ni recommandé car la taille de la piste est réduite de moitié. En plus, à cause de l'augmentation de la densité radiale (nombre de pistes) et linéaire (capacité de la piste), le support magnétique des disquettes 360 Ko et 1.2 Mo est différent : celui d'AT nécessite le champ magnétique plus fort que celui utilisé pour les disquettes « PC ». En fait, les deux médias sont virtuellement incompatibles et doivent être traités comme tels. Evidemment, cette différence de support et de format a crée beaucoup de la confusion et de difficultés pour les utilisateurs. Fort de l'expérience, IBM a décidé de ne pas récréer le chaos avec les disquettes de 3"1/2 et a rapproché davantage les standards double densité 720 Ko et haute densité 1.44 Mo : sur les deux, le nombre de pistes est le même (80). C'est pour cela que vous avez moins de problèmes à manipuler les disquettes 3"1/2 de standards différents. Mais le problème de la densité d'enregistrement persiste : le support magnétique des disquettes HD a besoin davantage du champ magnétique que celui de DD. Le champ magnétique nécessaire pour enregistrer une disquette DD est de 600 oersted tandis que celui utilisée pour enregistrer la disquette HD est de 700 oersted et le support magnétique (média) doit être adaptée. Vous pouvez tenter d'enregistrer les disquettes DD dans les lecteurs HD, mais l'inverse n'est pas vraie - le lecteur DD n'est pas simplement « assez fort » pour écrire sur les supports magnétiques HD. Si vous enregistrez en HD densité sur une disquette DD, vous risquez de rencontrer des problèmes et notamment la perte de l'information au bout d'un certain laps de temps. Bien que cela peut marcher pour des opérations courantes, du jour, nous ne pouvons pas vous le recommander pour des disquettes qui doivent garder les informations sur une longue période de temps (un an) ou pour des enregistrements vitaux (sauvegardes etc.).

Pour ne pas esquinter ni vous, ni la disquette en manipulant un couteau, il existe aussi des pinces qui servent à percer le trou supplémentaire dans le boîtier d'une disquette DD, ce qui trompe le lecteur et le fait enregistrer en HD. Voici les adresses de deux vendeurs de telles pinces, en USA (nous ne connaissons pas en Europe) :

CAJUN EDGE Inc.
117 S Oak ST Ste C, Dept I
Hammond, LA 70403
Téléphone : +1 (504) 542 0410
prix US $ 39.95-

COUNTRY CRAFT
PO Box 62
Norco, CA 91760
Télephone : +1 (714) 279 4360
prix : US $ 9.95-

DEBUG MAGAZINE

Rédacteur en Chef
Directeur de Publication
Artur G. Piotrowski

Maquettiste
Elisa Le Moal

Ont collaboré à ce numéro
ART, Laurent Chambaret, Serge Grun, Jean Christophe Riat, Andy Klein, Pierre - Morel Fourrier, Jean Baptiste Courtois, Alan Deikman, Frédéric Berton, Robert J. Moore.

Iconographie
Photos : Borland, PC Soft, Pictor International, Debug Magazine

REDACTION
TELEMATIQUE
VENTE AU NUMERO
SERVICE LECTURE
GESTION DES ABONNEMENTS
EDITEUR
SOFTMEDIA
SARL de presse au c[al] de 5000 F.
3, rue de l'Atlas
75019 Paris
Téléphone : (1) 42 02 29 16
Télécopie : (1) 42 45 58 13

Coordination Production
Christian Hernault

COMPTABILITE - ABONNEMENTS
Lidia Zakan

Prix de l'abonnement
11 numéros + 1 numéro spécial
486 F. T.T.C. (France)
Prix au numéro : 37 F.

IMPRESSION - BROCHAGE
La Presse de Bretagne

Mise en page : X.PRESS-Macintosh

Dépôt légal à la parution
Commission paritaire 71290
ISSN 0998-9250
membre inscrit à O.J.D.

« La loi du 11 Mars 1957 n'autorisant pas aux termes des alinéas 2 et 3 de l'article 41 d'une part que les copies ou reproductions strictement réservées à l'usage privée du copiste et non destinés à une utilisation collective et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, faites sans le consentement de son auteur ou de ses ayant droit ou ayant cause, est illicite (alinéa premier de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du code pénal.

SOMMAIRE

N° 12 - Mai 1991

EDITO 1

RUMEURS 6

EXCLUSIF : Turbo Pascal pour Windows 9

DOSSIER Dr. DOBB'S
Gestion de Mémoire sous MS - DOS 13
Gestionnaire de mémoire cache 19
Gestionnaire de périphériques MS - DOS en C 50

PROGRAMMATION
Protect et Vérif 26
Cahier Listing 29

INITIATION
Des arbres programmatiques pour une analyse naturelle 56
Notion d'algorithmique 63

JEUX
Génération automatique de grilles de mots casés 66

EVALUATION
Paradox 68
High Screen 70

PODIUM 54
DISQUETTES DU MOIS 62
IDEE 75
ANNONCES 78

LA PAGE DE JOHN C. DVORAK 80

RUMEURS

par *ART*

* La période actuelle de comptes et de prévisions abonde en annonces, bonnes et mauvaises nouvelles se suivant.

D'abord, l'histoire de « l'informatique française », qui semble approcher de son terme - l'industrie est rappelée à la dure réalité économique. Vous devinez de qui je parle : d'une part, de notre Bull national, le géant aux pieds d'argile et dont l'histoire n'est qu'une suite de lamentables faillites, et de l'autre, SMT Goupil, une (relativement) petite entreprise, qui s'est courageusement battue mais qui, à son tour, succombe aussi aux difficultés de la concurrence.

Quelques 7 milliards de francs qu'il va falloir à nouveau payer de la poche trouée du contribuable à Bull (et ils ne vont pas suffire), s'ajoutent à des dizaines de milliards déjà payés - certains estiment les investissements de l'État à quelque 100 milliards de francs au total. Les vrais comptes ne seront, vraisemblablement, jamais connus. Ce qu'on sait par contre, c'est qu'un industriel aux yeux bridés héritera : l'entrée de NEC dans le capital de Bull est prévue « avant l'été ». Le rêve est terminé? Non, le « ni-ni » continue. D'abord, on parlait de 10% de participation NEC. Maintenant, pour ne pas choquer certains beaux esprits, il n'y aura que 5%. Visiblement, les milliards enfouis à perte choquent moins que certaines idées reçues. Quand enfin ces fausses idées sur « l'informatique française » ne seront-elles qu'un mauvais souvenir? Certains parlent de l'abandon de toute activité de fabrication de Bull, au profit du seul développement des logiciels.

Détail croustillant : c'est ICL, récemment acquis par Fudjitsu, qui s'oppose à Bruxelles à l'attribution des nouvelles subventions de l'État à Bull. Outre-Manche, on oublie rapidement les 2 milliards de francs de subvention payés en 1981 par le gouvernement britannique à ICL. La firme anglo-nippone prends ainsi sa revanche, se sentant mal-aimée, écartée des projets européens depuis son achat par les japonais. Et pendant que nous, les européens, nous nous battons, chacun dans son coin, les autres ne laissent pas quartier libre...

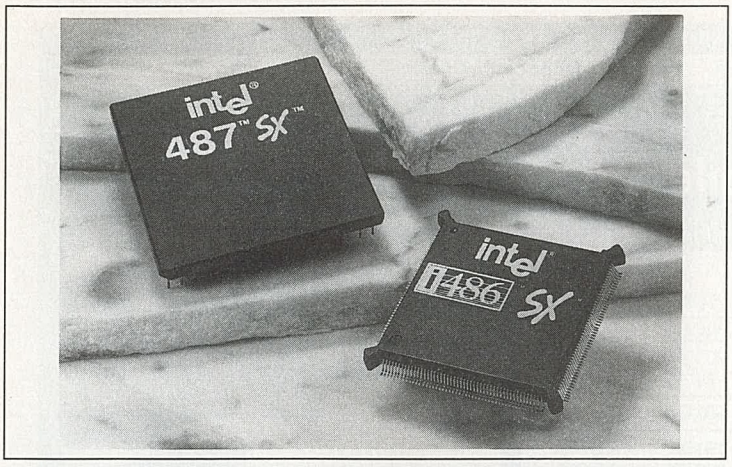
* L'histoire de Goupil est différente. Bien que longtemps protégée par l'administration de toute concurrence étrangère, notamment par de généreux contrats des PTT ou de l'Éducation Nationale, systématiquement accordés lors des achats publics malgré les prix aberrants de ses machines, l'entreprise se portait plutôt bien, au moins jusqu'à l'année passée. Mais toute fiction a une fin : encore une fois, le marché concurrentiel n'a pas toléré cette vie dans les nuages. Les prix des machines baissent systématiquement pendant que les prix des processeurs se maintiennent : la tenaille se referme lentement mais irrévocablement et seuls les gains de productivité en volume et l'efficacité commerciale assurent la survie face à la concurrence. Le château de cartes s'est disloqué durant le dernier exercice. Le chiffre d'affaire consolidé prévu était de 1.8 milliards; Goupil n'a réalisé que 830 millions. Le trou atteint 200 millions de pertes, au lieu de 30 millions de bénéfices prévus. Pour gagner du temps, l'entreprise « joue la montre », prolonge l'exercice comptable, étant virtuellement depuis quelques semaines en état de dépôt de bilan. Du temps pour élaborer un plan de redressement, pour trouver de nouveaux partenaires? Encore une fois, c'est l'État qui est au chevet, étant lui-même actionnaire par Crédit Lyonnais (5%) et France Télécom (17%) interposés. Cette fois-ci, le taiwanais Acer a été écarté, l'administration jugeant l'origine de ses fonds douteuse. On regarde du coté de Siemens-Nixdorf, mais à Munich on fait la sourde oreille; on envisage un rapprochement avec Bull; en tout cas la solution devra être trouvée rapidement car, à défaut, c'est un syndic qui devra s'occuper des créances en souffrance. La situation n'est pas aussi grave chez La Commande Électronique, mais tout de même. Cette entreprise vient justement de publier les résultats de l'exercice 1990, accusant une perte de 11 millions de francs. Décidément, ceux qui se frottent à Ashton Tate, y laissent des plumes... Pour redresser la situation, LCE compte sur les ventes de dBase, notamment la version UNIX (vous en avez certainement besoin en quantité...) et sur la progression des ventes de ses produits télématiques. Compte tenu des problèmes que LCE a avec PNB, qui l'accuse de piratage de ses modems, et la rupture de contrat avec GOTO, son fournisseur de logiciels, on peut en douter.

* Pour réconforter ce cocorico national un peu enroué, on pointe sur IBM qui annonce, pour la première fois dans son histoire, une perte de 10 milliards de francs pour le premier trimestre 1991. Du coté d'Armonk on l'impute sur le compte de la guerre du Golfe et « d'un climat incertain et difficile d'affaires ». Grand Bleu n'est cependant pas très fier. On s'empresse d'affirmer que « la Compagnie est en excellente position pour tirer parti de toute amélioration de l'économie », notamment grâce à la restructuration de ses revenus, qui proviennent à 45% d'autres activités que les ventes des matériels.

L'avenir d'IBM n'est pas si certain. La crise est généralisée. L'industrie américaine entend à se défendre, en créant une sainte alliance contre IBM et SUN, au sein d'un consortium baptisé Advanced Computing Environnement (ACE) et composé de 21 entreprises, notamment Microsoft, Compaq, DEC, Silicon Graphics, Santa Cruz Operation, NEC, Tandem, Acer, Sony, Wang ou Olivetti, lesquels décident soudainement de coopérer autour des processeur R4000 RISC de MIPS Computer Systems, processeurs qu'il va falloir, par ailleurs, attendre encore quelques années. L'objectif est d'offrir l'alternative aux processeurs d'INTEL (qui ne s'est pas joint au consortium), utilisés par IBM. Visiblement, Microsoft n'entend plus vivre dans l'ombre d'IBM, avec lequel ses dissensions ont fait l'objet d'un large écho ces derniers temps. Je parie qu'on va nous sortir à nouveau la version « portable » d'OS/2!

* INTEL se défend. Le 22 avril à Paris, le « géant de la puce » annonçait une nouvelle version de son processeur i486,baptisé INTEL486SX. Ce « nouveau »processeur est présenté

[SERVICE LECTEURS : INTERESSANT = 3 / MOYEN = 4 / MAUVAIS = 5]



Nouveau processeur INTEL : 487 SX

comme une version économique d'INTEL486DX : d'entrée de gamme, un « i486 au prix d'un 386 ». Il est vendu au prix de 1519 F. (par 1000), ce qui est un tiers par rapport au « vra i» i486DX. La nouvelle puce fonctionne à la vitesse de 20 MHz et se distingue de son grand frère par l'absence du coprocesseur intégré. Un coprocesseur mathématique « externe » est d'ailleurs aussi présenté: l'INTEL487SX est vendu lui-même à 4700 F. l'unité, prix de détail.

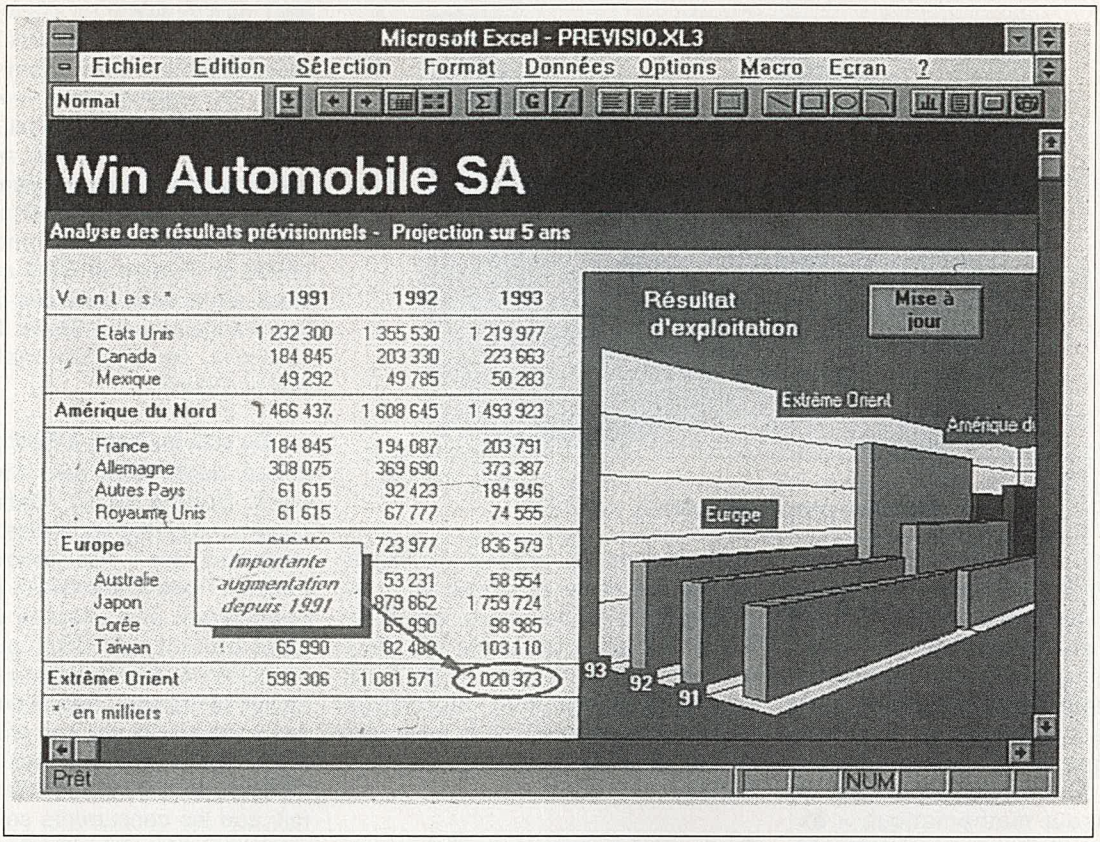
Une nouveauté ? Non. Cette fois-ci, c'est un pur « blitz » commercial. Qu'est, en fait, le i486SX? On pense qu'il s'agit tout simplement des puces qui ont failli soit aux tests de vitesse, soit aux tests de l'unité mathématique. On « tue »la connexion avec le coprocesseur, on re-carosse tout et on le vend en public. Et le i487SX? C'est encore plus fort: lorsque vous allez l'insérer dans l'emplacement prévu pour le coprocesseur, il supplante le i486SX dans toutes ses fonctionnalités-CPU et coprocesseur. Le i487SX se met à travailler à la fois en tant qu'unité centrale et que coprocesseur. Il s'agit donc d'i486DX complets qui ont « seulement » failli aux tests de la vitesse et dont l'unité mathématique est bonne!

Evidemment, INTEL se défend en disant que la série i486/7SX est fabriquée sur des lignes dédiées et que l'intégration de la puce est différente. Peu importe : qu'on la prélève dans les rejets au bout de la chaîne de fabrication ou qu'au départ on baisse les critères des tests, le résultat est le même: on nous vend des i486 identiques, à des prix différents. Car, à la différence du 386SX, le 486SX n'a pas de bus 16 bits (et INTEL jure qu'il ne produira jamais un tel bâtard), ce qui oblige à l'intégrer dans des architectures 32 bits natives. La seule économie pour les constructeurs (et pour nous, les utilisateurs) tient en la baisse de prix de la puce elle même, ce qui n'a rien à voir avec la différence entre l'architecture 16 bits d'un ordinateur à la base d'un 386SX et l'architecture 32 bits d'un 386DX.

On ne va plus vanter la qualité de processeur i486 : ses remarquables performances ne résultent pas simplement de la vitesse de l'horloge. Elles sont essentiellement dues à son architecture interne, à son niveau d'intégration, au décodage des instructions en parallèle, à l'exécution des principales instructions en un seul cycle horloge, à l'optimimalisation selon les principes RISC et à l'emploi d'un « cache » intégré, permettant de travailler avec les mémoires lentes (d'ailleurs INTEL assure ne jamais sortir des 486 sans « cache » ce qui résulterait en une significative dégradation des performances). Dans ce contexte, l'introduction de « nouveau »processeur i486SX ne peut être vue que comme une contre-attaque envers les « cloneurs » (qu'INTEL appelle d'ailleurs des copieurs) et notamment envers AMD qui vient annoncer les livraisons de son AMD386 à 40 MHz, retenu d'emblée par plusieurs fabricants d'ordinateurs. INTEL veut, tout simplement, tuer le marché des 386 (il n'y aura pas de versions plus rapides que 33 MHz), au profit du processeur plus performant, dont il a seul le secret. INTEL486SX est simplement l'enfant de la concurrence...

* Et quoi ensuite, pardon, quoi de NeXT? Eh bien, on s'installe en France! Steve Jobs, qui n'a visiblement pas froid aux yeux, s'installe à Sophia-Antipolis (il est vrai qu'il fait chaud, là-bas), où il va établir un siège européen de 1800 m^2, dédié à la recherche et au développement (lire : commerce) et créer 300 emplois. Jobs veut « reproduire avec NeXT ce qui a fait le succès d'Apple », sortir des nouvelles machines de « seconde génération », et il pense vendre cette année 50000 unités dans le monde entier, dont 15000 en Europe et 5000 en France. Nous allons enfin avoir des cubes parlant français, merci Steve! A propos, celui-ci là, ne voudrait-il pas reprendre Bull ou Goupil?

* Du coté du Nord, à Redmond, on a chaud, aussi, mais pour d'autres raisons. Depuis un an, Gates et ses copains reçoivent les visites de tristes messieurs, des « incorruptibles » de la Federal Trade Commission (FTC), qui poursuivent une enquête anti-trust, sujet brûlant aux USA, pays de la concurrence. En reproche à la joyeuse équipe de tromper, depuis des années, ses concurrents sur ses vraies intentions, faisant croire à tous que c'est OS/2 qui sera le système d'avenir. Une fois que les concurrents se sont tous engagés dans les développements OS/2, on aurait sorti WINDOWS et les applications qu'on a mijotées en cachette. En plus, Microsoft ne livrerait pas des librairies stables ni de documentation complète permettant aux autres de développer facilement et rapidement des applications. On dit aussi que, dès qu'un concurrent potentiel annonce une nouvelle idée ou un produit, Microsoft se livrerait à la désinformation, se précipitant avec un prototype bâclé juste pour détourner l'attention de l'annonce concurrente. On l'accuse de monopoliser le marché des logiciels, contrôlant d'une part les systèmes opérationnels, d'autre part les applications. Les développeurs soulignent que, par exemple, certaines techniques de manipulation d'objets, développées pour EXCEL 2.0 ou OLE, intégrées déjà il y a un an à Powerpoint et toujours non livrées aux développeurs, ont été réutilisées dans WINDOWS 3. Monsieur Gates, personnellement, superviserait tant les lignes du code d'applications que celles du code des systèmes opérationnels. L'enquête prend de l'envergure : il y a une semaine, FTC a notifié à Microsoft l'élargissement de l'investigation : on va voir si les « bundles » du DOS et WINDOWS avec certaines machines ne donnent pas à Microsoft l'avantage pour vendre ensuite ses applications. Serait-ce vrai? Nooon! Dans un interview à USA TODAY (26/3/91), Billy Boy dément catégori-



Microsoft Excel 3.0

quement ces allégations, les qualifiant de « mensonges directs ». « Ce n'est qu'une jalousie. C'est juste le fait que les gens n'aiment pas une compagnie qui a autant de succès que la nôtre ». Parlant de ses détracteurs (dont on ne connaît, évidement, aucun nom), il dit : « vous pouvez pendant longtemps assassiner les gens en coulisse. Mais cette fois-ci, quelqu'un va sortir avec les mains sanglantes, car c'est une fantaisie, pure invention ». Dans un autre interview, William Neukon, vice-président des affaires juridiques de Microsoft, nous éclaircit sur ces quelques-uns: « Nous connaissons quelques personnes, comme Jim Manzi (président de Lotus Development Corp), qui pensent que nous nous sommes payés leur tête en les poussant à écrire pour OS/2, et ce n'est pas le fait ». Il est vrai que ce « pauvre » Lotus a investi quelque 300 millions de francs dans le développement pour OS/2 et que l'investissement a du mal à se rentabiliser compte tenu de l'échec d'OS/2 et du phénoménal succès de WINDOWS.

Justement, le 2 avril Microsoft a présenté EXCEL 3.0. Je laisse à mes confères le soin de vous décrire en détail ce nouveau tableur, ergonomique et convivial, qui tire de WINDOWS le maximum de ses avantages, alliant dans un même ensemble les fonctions classiques d'un tableur, d'un traitement de texte, d'un gestionnaire de données, du dessin assisté et d'un outil de présentation. EXCEL 3.0 apporte une multitude des nouvelles fonctionnalités, comme le bouton de somme automatique, 8 niveaux de détail, la mise en forme, le positionnement automatique et l'importation des objets graphiques. Ses graphiques en 3D sont impressionnants : on peut modifier l'angle de vue par la rotation sur l'axe, en altitude et en perspective. Il sait exploiter des fichiers d'autres formats, comme WKS, SYLK et DBF, et communiquer dynamiquement avec des applications WINDOWS par DDE et OLE. On y trouve un « solveur »assisté qui résout des équations pour simuler des résultats demandés et, ultime perfection, un « module d'aide Lotus amélioré » (!). A ma question sur la chance que Microsoft laisse à des développeurs indépendants, voulant se lancer dans la conception d'un nouveau tableur, Michel Lacombe, président de Microsoft France, d'habitude réservé, a simplement répondu : « Aucune, maintenant c'est une industrie lourde ».

Enfin, revenant à l'affaire de FTC, Microsoft peut se réconforter : elle n'est pas toute seule. IBM est dans le coup : on lui demande aussi des explications. Et depuis quelques temps, Microsoft est devenue très gentille. Par exemple, Borland nous confirme une attitude « très coopérative » des gens de Redmond et souligne l'aide obtenue dans le développement de ses applications pour WINDOWS (comme Object Vision ou langages). « Ils veulent vraiment qu'on développe pour WINDOWS, ils nous ont passé presque tous les informations demandées ». Surprenant, non?

* Justement, Borland : pour terminer, une bonne nouvelle. Philippe Kahn va pouvoir s'acheter un nouveau saxo, cette fois-ci en or. Borland vient d'enregistrer auprès de la Securities and Exchange Commission (SEC) une offre publique de vente de 1.000.000 actions ordinaires. 900.000 nouvelles actions vont être vendues par la Société elle même et 100.000« par l'un de ses actionnaires ». Devinez qui...

A tout hasard, si vous êtes intéressé par ces actions (et c'est une bonne affaire) ou par les détails de l'offre (les noms!), écrivez à Lazard Frères & Co, 1 Rockefeller Plaza, New York, NY 10020. N'oubliez-pas de me mentionner, peut-être vont-ils m'offrir une action gratuite!

Turbo Pascal pour Windows

par Serge Grun

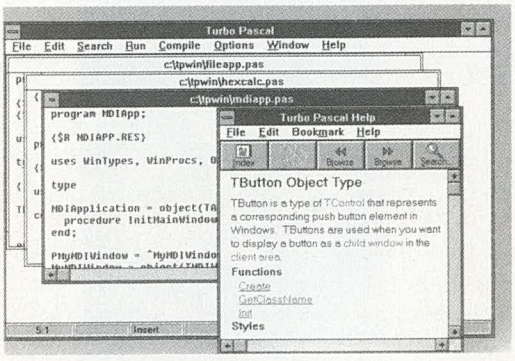
Vous, pour qui Begin et End n'ont plus de secrets, vous qui êtes des virtuoses du WriteLn et qui jonglez avec les Units alors que d'autres s'échinent à chercher stdio.h, vous regardiez Windows avec envie.
La programmation d'applications pour cet environnement graphique vous était jusqu'à présent interdite, à moins de recourir à l'assembleur ou, suprême horreur, au C!
Mais tout vient à point à qui sait attendre.
Allez donc examiner le bout de listing qui accompagne ces lignes.
Allez-y et prenez tout votre temps - je vous attendrai.

Ça y est ? Vous avez reconnu le code ? Oui, c'est du Pascal. Oui, c'est compilé par TurboPascal. Oui, ça tourne (et même très bien). Oui, ça tourne sous Windows 3.0. Et non seulement que ça tourne sous Windows, mais en plus, c'est une application Windows à part entière, avec sa fenêtre, son menu, ses icônes et tout le reste.

Vous l'avez deviné, Turbo Pascal for Windows (TPW) vient de paraître. Trois disquettes haute densité et sept manuels apportent un environnement de développement complet. Contrairement à Borland C++, TPW est une application Windows. C'est la transposition, ou plutôt l'adaptation, de l'environnement intégré de développement des précédentes versions de Turbo Pascal à l'environnement graphique de Windows.

Développer Pour Windows ... Sous Windows

Cette application rassemble à un éditeur multi-fichiers, conforme aux spécifications MDI (Multiple Document Interface) de Windows et contient un compilateur Pascal qui génère du code natif, permettant de créer des exécutables Windows, des Units, l'échange dynamique de données (DDE) et des librairies dynamiques (DLL). Les utilisateurs des versions précédentes ne seront pas dépaysés - l'éditeur reconnaît l'ensemble des commandes reconnues par les versions DOS. Il accepte également le jeu de commandes CUA/SAA (recommandé par IBM), le choix peut être modifié à tout moment et prend effet immédiatement. De plus, l'éditeur est entièrement reconfigurable à l'aide d'un macro-language.



Environnement intégré de développement de Turbo Pascal Windows et la fenêtre de l'aide contextuelle.

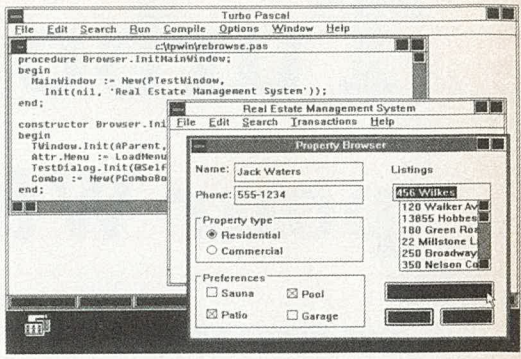
Compilation Transparente

La partie compilateur est tout à fait comparable à celle des versions DOS. L'on retrouve la plupart des options de compilation, à l'exception de celles qui ne s'appliquent pas à Windows (notamment les besoins en mémoire qui sont remplacées par la définition de la taille du tas et de la pile). Une option est nouvelle: c'est la génération des prologues et des épilogues dans les fonctions exportables. Cette génération n'est nécessaire que pour les programmes qui doivent fonctionner dans le mode réel de Windows et elle peut être désactivée pour la compilation d'une application qui ne fonctionnera qu'en mode standard ou 386 étendu.

Le compilateur assure la génération du code, l'édition des liens et le rattachement des ressources, le tout d'une façon totalement transparente, qu'il s'agisse d'un exécutable ou d'une librairie dynamique. La notion d'Units n'a pas été abandonnée: il est possible d'en créer et même d'y inclure des ressources. Pour des raisons propres à Windows, la compilation aboutit toujours (sauf erreur de syntaxe) à la création d'un fichier sur disque et il n'y a plus de possibilité de compiler en mémoire.

Sur L'autoroute Borland, La Vitesse N'est Pas Limitée !

Notre compilateur a été installé sur une machine à base de 386SX à 20 MHz avec 5.5 MB de mémoire vive. L'installation de la totalité des fichiers nécessite environ 6.5 MB de l'espace sur disque. TPW s'exécute en mode protégé uniquement. Dans cette configuration, Windows en mode standard laisse plus de 4 MB de mémoire, ce qui permet de placer le tampon d'édition des liens en RAM. La vitesse de compilation est alors impressionnante. Ne disposant pas d'applications de grande taille, je n'ai compilé que les fichiers exemples fournis avec TPW: j'ai constaté que les temps de compilation étaient trop courts pour être mesurés de façon fiable! A titre indicatif, sur des fichiers de taille semblable, TPW demande entre 5 et 30% de temps en moins que Borland C++ 2.0 et jusqu'à 50% en moins que le Zortech C++ 2.1. Borland affirme que TPW compile plus de 85000 lignes par minute (sur un Compaq Deskpro 386/33).



Avec Oject Windows, vous pouvez rapidement créer des applications Windows rafinées.

Le programme compilé peut aussitôt être testé, puisqu'il n'est plus nécessaire de quitter le compilateur pour charger Windows. Inutile de préciser que le gain de temps et de productivité, ainsi que le confort d'utilisation obtenus sont particulièrement importants.

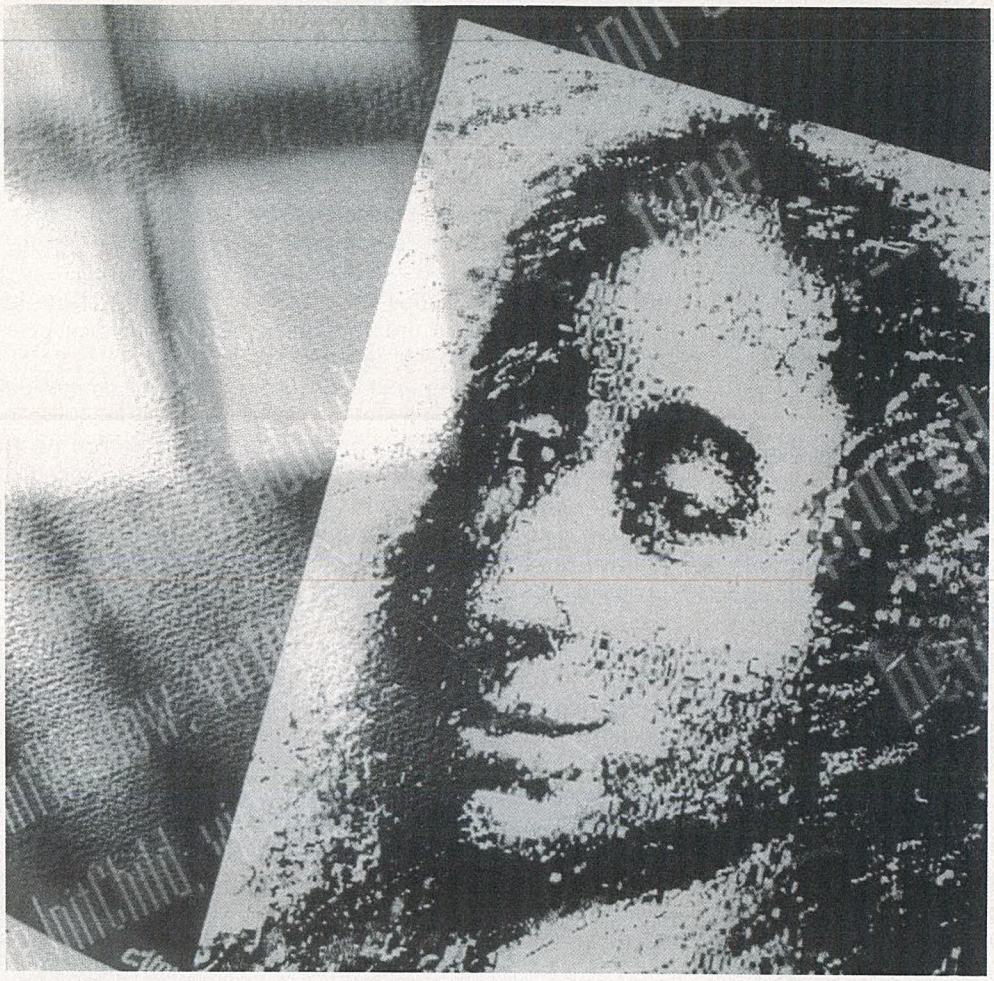
Si je n'ai pas, jusqu'à maintenant, parlé du débogueur, c'est parce que l'environnement intégré en est dépourvu. Cependant, le débogueur ne manque pas pour autant. TPW est livré avec TurboDebugger for Windows 2.5 (TDW). TDW est une application un peu bâtarde, en ce qu'il s'agit d'un exécutable Windows, mais qui fonctionne en mode texte. L'interface utilisateur reproduit, à quelques détails près, l'aspect de l'interface de la version DOS - la nouveauté majeure étant la possibilité de déboguer des applications Windows et l'adjonction d'une fenêtre supplémentaire, permettant d'intercepter les messages que Windows adresse à l'application.

Le débogueur peut être invoqué à partir de l'environnement intégré. Il permet de déboguer aussi bien les exécutables que les librairies dynamiques. Ses besoins matériels sont les mêmes que ceux de TPW (80286, 2 MB de mémoire vive et Windows 3.0 en mode standard ou 386 étendu).

Trois Méthodes De Développement

L'intérêt majeur de cette nouvelle version c'est évidemment de pouvoir développer des applications Windows sans le Kit de Développement Windows (SDK) de Microsoft et, si possible, avec le minimum d'efforts. TPW a l'originalité de proposer trois méthodes différentes de développe-

ment qui permettent au programmeur de progresser à son propre rythme dans l'apprentissage de la programmation pour Win-dows.



La première méthode n'a rien d'original. Elle consiste à utiliser l'unité standard WinCrt. Celle-ci offre la compatibilité avec toutes les fonctions d'entrée/sortie standard de Tur-bo Pascal. Les applications créées de cette façon fonctionnent dans une fe-nêtre mais se comportent comme des programmes DOS. L'affichage se fait comme sur un terminal en mode texte, vers un écran virtuel de 25 lignes et 80 colonnes, en utilisant un jeu de caractères à espacement fixe. Si vous dépassez les limites de la fenêtre, celle-ci défile automatiquement et le contenu précédent est perdu. L'utilisation de GotoXY reste permise, rendant possible l'adaptation de programmes qui fonctionnent sous DOS, sous réserve qu'ils se bornent à fonctionner en mode texte. L'utilisation des éléments de l'interface utilisateur de Windows ou l'accès aux primitives graphiques exigent alors des efforts surhumains. L'avantage est que c'est facile et ça marche (mais je ne suis pas certain qu'il y ait un intérêt quelconque).

Une autre possibilité de développement consiste en la méthode traditionnelle, telle qu'elle est décrite dans le Kit de Développement Windows. Cette technique consiste à écrire, ligne par ligne et fonction par fonction, tout le code nécessaire pour que l'application puisse créer et enregistrer les classes, créer et afficher les fenêtres et répondre aux quelque 200 messages que Windows est susceptible de lui adresser. L'équivalent le plus proche serait de s'engager sur une galère, de s'asseoir sur le banc de nage d'un roprogrammeur C et à ramer en cadence. L'avantage est que le cycle développement - compilation - essais est considérablement raccourci et qu'il est possible à tout moment d'accéder à toutes les fonctions de Windows (notamment aux éléments de l'interface et aux fonctions graphiques).

Object Windows - La Librairie Réutilisable

J'ai gardé le meilleur pour la fin. Les versions DOS de Turbo Pascal possèdent des extensions objet que l'on retrouve dans TPW. Dans Turbo Pascal 6.0 il y a même une hiérarchie d'objets, appelée TurboVision, qui fonctionne selon un modèle dit 'event driven' et qui permet de créer, avec un minimum d'efforts, des interfaces utilisateurs comparables à celles des environnements intègres des versions DOS. Le modèle 'event driven' est remarquablement bien adapté à la structure interne de Windows puisqu'il fonctionne exactement de la même façon: des objets répondant à des messages. Quoi de plus naturel alors que de créer une hiérarchie d'objets qui reproduisent, à l'aide d'ob-jets de Win-dows, l'équivalent de Turbo Vision? C'est ce qui a été fait: TPW est livré avec une hiérarchie appelée Object Windows. On retrouve dans cette hiérarchie tous les éléments constitutifs de Windows : l'application, avec la fonction WinMain, les fenêtres, avec leurs fonctions de gestion associées, les dialogues, les boutons etc. Tous ces éléments dérivent d'un objet ancêtre commun dont ils héritent le comportement. Les méthodes initiales se chargent de créer ces objets avec des valeurs par défaut et de répondre raisonnablement à la plupart des messages.

La création d'une application devient alors aussi simple que celle illustrée par le listing: quelques lignes à peine pour initialiser une application, créer et afficher la fenêtre principale et entrer dans la boucle d'attente de messages. La création de nouvelles fenêtres ou dialogues se fait par dérivation des objets existants et la surcharge de quelques fonctions. Il n'est plus nécessaire d'écrire des pages entières d'instructions CASE pour que l'application puisse réagir aux messages envoyés par les menus: il suffit d'écrire une fonction et de préciser pour quel type de message elle doit être invoquée.

A titre indicatif, l'écriture, en C, d'un bête programme qui lit un fichier d'entiers et qui les affiche sous la forme d'un graphique, m'a demandé quatre jours. Pourtant, la programmation Windows ne m'est pas étrangère et j'ai construit l'application à partir d'un squelette de base relativement complet. L'écriture d'une application, comportant exactement les mêmes

fonctionnalités à l'aide d'Object Windows, a demandée six heures de travail (si, si, vous avez bien lu!) et je n'ai pas eu à utiliser une seule fois le débogueur. Object Windows est indéniablement l'outil le plus contributif dans la création d'une application Windows, d'autant plus que cette encapsulation ne gêne en rien l'accès aux détails de Windows: il suffit de sur charger une méthode ou de redéfinir une valeur par défaut, pour se retrouver de nouveau aux commandes.

Whitewater Ressource Toolkit

Ceux qui se sont déjà frottés à la programmation Windows savent que le compilateur seul est insuffisant et que plusieurs autres outils sont nécessaires, comme notamment un compilateur de ressources ou un éditeur d'icônes. Ce problème est résolu de façon élégante, puisque avec chaque exemplaire de TPW Borland livre le Whitewater Ressource Toolkit (WRT). The Whitewater Group est une entreprise américaine qui vend de nombreux outils de développement pour l'environnement Windows. Le WRT est la réunion en un seul produit d'un compilateur de ressources et de plusieurs éditeurs d'icônes, de curseurs, de dialogues, de menus, d'accélérateurs et de chaînes de caractères. Il a la capacité de créer des ressources, mais également d'éditer des ressources préexistantes qui peuvent indifféremment provenir de fichiers .RC (texte), .RES (ressources compilées), exécutables ou librairies dynamiques, permettant notamment de modifier des ressources dont l'on ne possède pas le fichier source. Grâce au WRT, le TPW permet de créer des applications Windows sans avoir recours au Kit de Développement Windows (SDK) de Microsoft, réalisant de ce fait une sérieuse économie.

Documentation

Cette présentation serait incomplète sans un mot sur la documentation. Les sept manuels couvrent quasiment tout ce qu'il est nécessaire de savoir pour utiliser pleinement le TPW. Ils se situent entièrement dans la lignée des manuels Borland. Deux d'entre eux sont consacrés au WRT et au TDW. Le troisième décrit l'utilisation de HC.EXE(un compilateur de textes d'aide contextuelle utilisables par WINHELP). Un Guide de l'Utilisateur (User's Guide, puisque la version examinée est américaine) couvre l'installation et l'utilisation de l'environnement intégré. Le Guide du Programmeur décrit la syntaxe du langage, les extensions objets, l'assembleur intégré, les détails internes du compilateur et les fonctions de la librairie standard. Il diffère relativement peu de son équivalent DOS (sauf en ce qui concerne les détails internes). Enfin, deux épais manuels décrivent la programmation Windows. L'un d'eux est le Windows Reference Guide. Il reprend la description des types, constantes et fonctions de l'API Windows ainsi que des objets et méthodes d'Object Windows (plus de 500 pages...). Par contre, la description des fonctions Windows est malheureusement trop concise (n'oubliez pas que le SDK de Microsoft n'est pas livré et c'est l'intérêt de la démarche!). Il vaut mieux consulter l'aide contextuelle de TPW (qu'on évoque dans une fenêtre) qui contient en plus des exemples d'utilisation. L'autre manuel est le Windows Programming Guide et il couvre l'utilisation d'Object Windows. Ce dernier manuel est particulièrement complet et décrit notamment la création de librairies dynamiques, l'échange dynamique de données, la gestion de la mémoire et la création d'applications avec documents multiples. Par contre, la programmation selon les techniques classiques n'est décrite que très succinctement - il est souhaitable de se procurer le « Guide to Program-ming » du Kit de Développement Microsoft (il peut être acquis séparément).

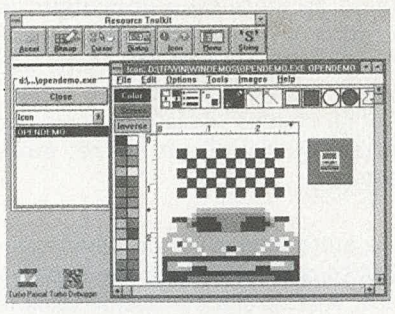
La documentation sur disque comporte essentiellement des corrections des manuels. L'exception la plus flagrante est la documentation du compilateur de macros pour l'éditeur. Depuis l'introduction du nouvel environnement de développement (avec Turbo C++), cette documentation est systématiquement bâclée dans un fichier traînant au fond d'une disquette, alors qu'il serait si simple d'ajouter une dizaine de pages qu'il représente à la fin du Guide de l'Utilisateur (par exemple...).

Un Produit Exceptionnel

A part quelques couacs dans la documentation, TPW est un produit exceptionnel. Il l'est, d'abord, en remplissant un créneau laissé vacant par Microsoft, à savoir celui de la programmation Pascal (le Kit de Développement de Windows 3.0 ne comporte plus les fichiers nécessaires pour le Pascal). Il l'est aussi parce qu'il permet de développer dans l'environnement cible (et agréable - Windows!), en raccourcissant notablement le cycle développement-tests.

Il l'est surtout parce qu'à la vitesse de compilation il allie la facilité de programmation, grâce à une hiérarchie d'objets qui renvoie au musée une bonne partie de la complexité de la programmation pour Windows.

TPW ne permet pas une optimisation des applications aussi poussée que celle que l'on peut obtenir en utilisant le C. Les exécutables sont également un peu plus gros (bien que l'éditeur de liens élimine les méthodes inutilisées) - ces deux arguments fourniront sans doute de l'eau au moulin de ceux qui prétendent que l'on ne peut pas développer d'applications professionnelles avec TPW. Mais, qu'est ce qu'une application professionnelle: serait ce une application dont les bugs sont commis par des programmeurs professionnels ? Malheureusement pour eux, ils ne trouveront guère que ces deux arguments, dont le poids est bien faible compte-tenu de l'équipement moyen qui accueille Windows 3.0. Quoi qu'on en dise, il faut reconnaître que la création d'applications pour Windows souffre de l'obligation d'utiliser un langage obscur pour programmer dans un environnement particulièrement complexe. TPW avec Object Windows apporte aujourd'hui une solution à ce problème et l'on ne peut que s'en réjouir.



Ce produit n'est pas encore disponible en France, pour toutes informations, contactez :
BORLAND
43, av de l' Europe
BP 6 - 78141 Vélizy Cédex
Tél : 39.46.96.69

La Gestion Mémoire sous MS-DOS

par Robert J. Moore
Traduction par Pierre Morel-Fourrier

Connaître la manière dont DOS gère la mémoire de votre PC vous économisera du temps et des soucis.

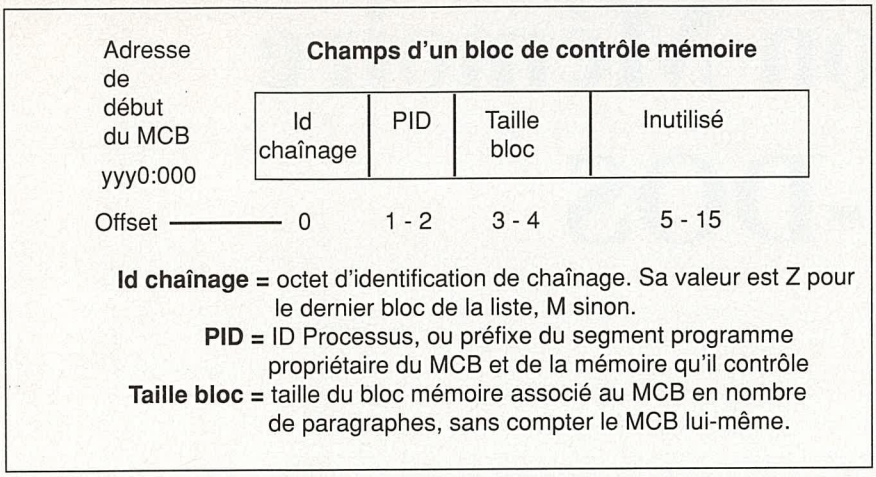
Le concept de bloc de contrôle de la mémoire (Memory Control Block, MCB) a été introduit à partir de la version 2.0 de MS-DOS, comme une méthode de base pour gérer la mémoire allouée par les applications et les gestionnaires de périphériques. Nous verrons dans cet article comment DOS gère ces blocs de contrôle de la mémoire et nous présenterons un programme en Turbo-C, appelé MCB.C, qui affiche l'état des allocations mémoire du DOS. D'abord, examinons comment le PC utilise sa mémoire.

La figure 1 représente la carte mémoire telle qu'elle est documentée par IBM et Microsoft. Les adresses sont indiquées en hexadécimal sous la forme segment:offset. Les adresses numériques indiquent les emplacements mémoires qui sont fixes depuis la version 2.0 de MS-DOS. Les adresses symboliques varient en fonction de la version du système et du nombre de programmes présents en mémoire. Il se peut en effet que plusieurs programmes soient présents simultanément en mémoire si des résidents ont été installés.

Adresse (Hex)	Utilisation de la mémoire
0000:0000	Table des vecteurs d'interruption
0040:0000	Zone de données ROM BIOS
0050:0000	Zone des paramètres DOS
0070:0000	IBMIO.COM/IO.SYS (1)
mmmm:mmmm	IBMDOS.COM/MSDOS.SYS (1)
mmmm:mmmm	CONFIG.SYS (gestionnaires de périphériques et tampons internes)
mmmm:mmmm	COMMAND.COM résidant
mmmm:mmmm	Environnement principal
mmmm:mmmm	Bloc d'environnement no 1
mmmm:mmmm	Programme no 1
... ...	
mmmm:mmmm	Bloc d'environnement no N
mmmm:mmmm	Programme no N
xxxx:xxxx	COMMAND.COM temporaire
A000:0000	Tampons vidéo et ROM
FFFF:000F	Haut de l'espace adressable 8086/88

(1) PC-DOS utilise IBMBIO.COM et IBMDOS.COM, MS-DOS utilise IO.SYS et MSDOS.SYS à la place.

Figure 1 : Carte mémoire du PC



Id chaînage = octet d'identification de chaînage. Sa valeur est Z pour le dernier bloc de la liste, M sinon.
PID = ID Processus, ou préfixe du segment programme propriétaire du MCB et de la mémoire qu'il contrôle
Taille bloc = taille du bloc mémoire associé au MCB en nombre de paragraphes, sans compter le MCB lui-même.

***Figure 2 :* Champs d'un MCB**

```
C> MCB
=============================================================
MCB MCB   ID PID     MB   PAR- ENV   OWNER
NO. SEG            SIZE ID   BLK?
=============================================================
01  0973 M  0008    8208 022B N     IBMDOS.COM/MSDOS.SYS
02  0B75 M  0B76    3376 0B76 N     COMMAND.COM COPY #1
03  0C49 M  0000      48 F000 N     FREE MEMORY CONTROL BLOCK
04  0C4D M  0B76     160 0B76 Y     COMMAND.COM COPY #1
05  0C58 M  0C5E      64 0B76 Y     C:\TURBOC\DEV\MCB.EXE
06  0C5D M  0C5E   71232 0B76 N     C:\TURBOC\DEV\MCB.EXE
07  1DC2 Z  0000 530384 F000 N     FREE MEMORY CONTROL BLOCK
=============================================================
```

***Exemple 1 :* Système initialisé sans CONFIG.SYS ni AUTOEXEC.BAT. Le programme MCB.EXE a été lancé depuis le répertoire C:\TURBOC\DEV où il se trouve, pour cet exemple**

Le programme MCB.C (listing un), calcule et affiche les adresses indiquées dans la figure sous la forme mmmm:mmmm, y compris la plupart des adresses inconnues de la carte mémoire. La seule adresse qu'il ne peut calculer est celle du début de la partie temporaire d'un COMMAND.COM. Cette adresse est notée xxxx:xxxx dans le shéma. MCB.C vous permet aussi de voir comment les fonctions d'allocation mémoire travaillent, et vous indiquera aussi où toute votre mémoire est partie la prochaine fois qu'un message fatidique du type « Plus de mémoire disponible » s'affichera. Vous pourrez ainsi repérer les résidants gourmands en mémoire. Et enfin, MCB.C vous montrera l'utilité des réallocations de mémoire. Mais pour l'instant, parlons des blocs de contrôle de la mémoire...

Les blocs de contrôle de la mémoire

Microsoft et IBM disent bien qu'ils utilisent des MCB, mais nulle part ils ne donnent des détails sur la manière dont ils les utilisent. Le contenu d'un MCB est décrit à la figure 2. Un MCB contient trois champs : l'identificateur de chaînage, l'identificateur de processus (PID), et la taille du bloc. Ces trois champs occupent les cinq premiers octets d'un MCB. Les onze autres octets sont inutilisés. Un MCB débute toujours au début d'un paragraphe (16 octets), et occupe exactement un paragraphe en mémoire. L'identificateur de chaînage indique s'il s'agit du dernier MCB dans la liste (valeur Z) ou s'il y en a d'autres (valeur M). Le bloc de mémoire contrôlé par le MCB suit immédiatement le MCB en mémoire. Il commence au début du paragraphe suivant. La taille du bloc mémoire en nombre de paragraphe est égale à la valeur du troisième champ du MCB. Par la suite, nous noterons MB un tel bloc mémoire, pour les distinguer des MCB. La paire suivante de MCB/MB débute juste après la paire précédente. Les MCB peuvent être considérés comme une liste chaînée en avant, et les MB qu'ils contrôlent, placés en sandwich entre eux. En temps normal, il n'y a pas d'espaces entre les MCB/MB. Si DOS détecte une anomalie dans cette chaîne, il affiche un message d'erreur et le système se bloque. Nous avons eu l'occasion d'altérer délibérément cette chaîne, mais le système ne s'en aperçoit pas toujours.

La première paire de MCB/MB allouée pour le système contient toujours les fichiers IBMDOS.COM/MSDOS.SYS. Le nombre de MCB alloués dépend du nombre de programmes résidants. En général, chaque programme chargé en mémoire (résident ou non) possède deux MCB. L'un reçoit une copie de l'environnement, et l'autre le code et les données du programme. Si un programme alloue de la mémoire, pour ses besoins internes par exemple, il peut posséder plus de deux MCB. Si un programme résidant libère son MCB d'environnement (comme certains le font), il peut ne posséder qu'un seul MCB.

Les fonctions DOS de gestion de mémoire

Depuis la version 2.0 de MS-DOS, toutes les versions du système contiennent des fonctions pour allouer de la mémoire (fonction 48H), libérer de la mémoire allouée (fonction 49H), ou réallouer de la mémoire (fonction 4AH). Aucune de ces fonctions ne gère directement les MCB, mais les MB associés. Les changements provoqués par ces fonctions se répercutent par contre sur les MCB, comme on peut le voir en utilisant le programme MCB.C entre deux allocations mémoire par exemple. De plus, à partir de la version 3.0 de MS-DOS, une fonction permet de modifier la stratégie d'allocation de la mémoire (fonction 58H). La stratégie par défaut utilisée par MS-DOS 3.0 est celle de MS-DOS 2.0. Les versions 3.0 et ultérieures possèdent en plus un mode de gestion plus efficace qu'il est possible d'utiliser via cette fonction.

Notez que le DOS (le COMMAND.COM en fait) utilise ces fonctions pour allouer automatiquement deux MCB chaque fois qu'un programme non résidant est lancé (par la fonction EXEC 4BH du DOS). La mémoire est libérée automatiquement lorsque le programme est terminé. Toute zone mémoire allouée au cours de l'exécution d'un programme doit être explicitement libérée, sinon la zone mémoire restera non libérée. Bien sûr, aucune zone de mémoire d'un programme résidant n'est libérée automatiquement.

Une description plus détaillée de ces fonctions de gestion mémoire vaudrait un article à elle seule. En utilisant

MCB.C, ces fonctions deviendront moins mystérieuses. Après avoir lu cet article et utilisé le programme, vous pourrez même gérer vous même vos allocations mémoire et court-circuiter le DOS, bien que cela ne soit pas le but de cet article.

Repérer les MCB en mémoire

Les deux MCB d'un programme considéré peuvent être repérés facilement. L'un d'entre eux se trouve un paragraphe avant le PSP (page de base du programme). L'autre se trouve un paragraphe avant le bloc d'environnement. L'adresse du segment où se trouve la copie de l'environnement se trouve à l'octet 2CH du PSP. Cela ne se vérifie malheureusement pas pour COMMAND.COM où, dans les versions 2.0 à 3.2, l'adresse du segment contenant l'environnement était à zéro. La version 3.3 corrige cette erreur.

Mais cela ne nous aide pas à trouver les MCB en mémoire basse car ceux-ci ne sont chaînés que vers l'avant. Il faut en fait repérer le premier MCB en mémoire. Ensuite il suffit de parcourir la chaîne pour trouver les autres MCB. Mais comment repérer le premier MCB de la chaîne ? Il n'y a pas de moyen documenté. Il existe un certain nombre de méthodes utilisant la force brute, mais aucune n'est vraiment satisfaisante. La moins mauvaise utilise la fonction non documentée Invars 52H. Quand cette fonction existe, ES:[BX-2] pointe sur un mot qui contient l'adresse de segment du premier MCB. Et comme tous les MCB commencent à une frontière de paragraphe, cela suffit pour localiser le premier MCB.

```
C>TYPE \CONFIG.SYS
device=c:\dos\ansi.sys
files=50
buffers=20
lastdrive=z

C>MCB

==========================================================================
MCB MCB   ID PID      MB   PAR- ENV  OWNER
NO. SEG             SIZE ID   BLK?
==========================================================================
01  0973 M  0008  16352 022B N    IBMDOS.COM/MSDOS.SYS
02  0D72 M  0D73   3376 0D73 N    COMMAND.COM COPY #1
03  0E46 M  0000     48 F000 N    FREE MEMORY CONTROL BLOCK
04  0E4A M  0D73    160 0D73 Y    COMMAND.COM COPY #1
05  0E55 M  0E5B     64 0D73 Y    C:\TURBOC\DEV\MCB.EXE
06  0E5A M  0E5B  71232 0D73 N    C:\TURBOC\DEV\MCB.EXE
07  1FBF Z  0000 522240 F000 N    FREE MEMORY CONTROL BLOCK
==========================================================================
```

Exemple 2 : **Système initialisé avec le CONFIG.SYS indiqué, et sans AUTOEXEC.BAT.**

MCB.C vous permet de voir comment les fonctions d'allocation mémoire travaillent, et vous indiquera aussi où toute votre mémoire est partie la prochaine fois qu'un message fatidique du type « Plus de mémoire disponible » s'affichera.

Des informations supplémentaires

Chaque paire MCB/MB est possédée par un PID, qui n'est autre que l'adresse du segment du PSP du programme propriétaire de la mémoire. A partir du PSP, on peut obtenir d'autres informations utiles. On trouve d'abord l'adresse de la copie de l'environnement, comme mentionné plus haut. Une autre information (non documentée) est la présence du PID du processus père. C'est un mot qui se trouve à l'adresse PSP:16H.

Et à partir de DOS 3.0, il est possible de récupérer le nom du programme propriétaire du MCB. Ce nom est composé du chemin d'accès complet et du nom du programme. Pour le trouver, localisez d'abord l'environnement d'un programme autre que COMMAND.COM ou IBMDOS.COM (ces programmes doivent être traités à part) en utilisant le mot se trouvant à l'adresse PID:2CH. L'environnement contient une suite de chaînes de caractères terminées par un caractère nul. Deux octets nuls consécutifs indiquent la fin des chaînes d'environnement. Si la valeur qui suit ces deux octets nuls est un mot égal à 0001, alors le nom du programme suit immédiatement. Si la fin de l'environnement est rencontrée et qu'aucun mot égal à 1 ne soit trouvé, le propriétaire du MCB est inconnu (et c'est toujours le cas avec MS-DOS 2.x). Notez que si un résidant a libéré son environnement, il n'est pas possible de déterminer le nom du propriétaire, car le bloc peut avoir été corrompu.

```
C>TYPE \AUTOEXEC.BAT
path c:\dos;c:\util;c:\turboc;c:\util\norton;c:\masm;c:\pe
prompt $p$g
mgc
astclock
subst u: c:\util
fastopen c:
cls

C>MCB
==========================================================================
MCB MCB   ID PID      MB   PAR- ENV  OWNER
NO. SEG             SIZE ID   BLK?
==========================================================================
01  0973 M  0008  16352 022B N    IBMDOS.COM/MSDOS.SYS
02  0D72 M  0D73   3376 0D73 N    COMMAND.COM COPY #1
03  0E46 M  0000     48 F000 N    FREE MEMORY CONTROL BLOCK
04  0E4A M  0D73    160 0D73 Y    COMMAND.COM COPY #1
05  0E55 M  0E5F    128 0D73 Y    C:\DOS\FASTOPEN.EXE
06  0E5E M  0E5F   2896 0D73 N    C:\DOS\FASTOPEN.EXE
07  0F14 M  0F1E    128 0D73 Y    C:\TURBOC\DEV\MCB.EXE
08  0F1D M  0F1E  71232 0D73 N    C:\TURBOC\DEV\MCB.EXE
09  2082 Z  0000 519120 F000 N    FREE MEMORY CONTROL BLOCK
==========================================================================
```

Exemple 3 : **Système initialisé avec le CONFIG.SYS de l'exemple 2 et l'AUTOEXEC.BAT indiqué.**

Le programme MCB

Le programme MCB (listing un) est largement commenté, nous ne ferons donc que quelques commentaires à son propos. Le compilateur utilisé est Turbo C, version 1.5, en modèle mémoire small. Ce modèle mémoire implique l'utilisation de pointeurs far ou huge, mais des commentaires expliquent cela. Aucune fonction spécifique à Turbo C n'est appelée, ce programme pourra donc être compilé facilement avec n'importe quel compilateur C.

Le programme lui-même est une application directe des informations décrites ci-dessus. Le premier MCB est localisé grâce à la fonction DOS 52H. Le programme explore chaque MCB/MB et en affiche les informations.

L'exemple 1 montre le résultat d'une exécution. C'est le nombre minimum de MCB que l'on peut avoir. Le MCB 03 semble avoir été utilisé par le DOS, puis libéré. Notez également la présence des MCB du programme MCB.EXE, et du grand MB à la fin représentant toute la mémoire libre. Lorsqu'une application est lancée, toute la mémoire lui est attribuée, et donc à cet instant, il n'y a pas de MCB en fin de liste représentant la mémoire libre. Mais à l'initialisation du programme, la fonction 4AH est appelée, et rend au système la mémoire superflue dont elle n'aura pas besoin, créant ainsi un MCB/MB représentant la mémoire libre.

Entre la figure 1 et la figure 2, la taille du MB 01 a augmenté de 8144 octets. Cette espace mémoire supplémentaire a été alloué pendant l'initialisation du système pour répondre aux instructions du fichier CONFIG.SYS. Ces instructions ont provoqué une réservation de mémoire supplémentaire pour l'installation du gestionnaire ANSI.SYS, ainsi que pour les instructions BUFFERS=20 et FILES=50 qui nécessitent plus de place mémoire. La taille de ce bloc augmente encore si, par exemple, VDISK.SYS est installé.

Les utilisations de ce programme sont multiples. Vous pouvez par exemple examiner en détail les effets des fonctions d'allocation mémoire.

Dans l'exemple 3, la commande FASTOPEN de DOS 3.3 apparaît sous la forme d'un programme résidant possédant deux blocs de mémoire. L'un des deux blocs est l'environnement du programme, l'autre bloc contient le code et les données du programme.

L'exemple 4 indique que la commande DOS GRAPHICS est également un programme résidant, et qu'elle possède également deux blocs de mémoire. Certains programmes résidants libèrent leur environnement avant d'exécuter la commande TSR DOS. De tels programmes n'auront donc plus d'environnement, mais en plus, le nom du propriétaire du programme qui se trouve dans l'environnement peut êtrealtéré, et ne pourra être affiché. Un programme résidant peut allouer de la mémoire supplémentaire pendant son initialisation. Ces blocs alloués seront repérés et affichés par MCB.EXE.

Dans l'exemple 5, une deuxième instance de COMMAND.COM possède trois blocs de mémoire. Le troisième bloc est alloué pour ses besoins propres.

L'exemple 6 montre que les blocs de mémoire réservés pour la deuxième instance de COMMAND.COM, sont effacés de la mémoire lorsque l'on tape la commande exit. S'il est possible de lancer plusieurs instances de COMMAND.COM, les blocs alloués par ces instances sont effacés dès que l'on quitte l'interpréteur de commande.

Dans l'exemple 7, CHKDSK indique une taille mémoire libre de 588.064 octets, alors que dans l'exemple 6, le MCB indique 516.816 octets libres dans le bloc mémoire associé. La différence est due à la présence en mémoire du programme MCB.EXE dans l'exemple 6. Cette différence est égale aux 71232 octets du programme MCB.EXE plus 16 octets de MCB. Et en fait cela est logique. Quand CHKDSK s'exécute, de la mémoire lui est allouée (en fait toute la mémoire disponible). Quand il affiche la taille mémoire disponible, il soustrait l'espace dont il a besoin mais pas l'espace de son environnement, pensant sans doute que tout programme lancé aura besoin de cet environnement. Notez aussi que CHKDSK ne tient pas compte des blocs libres en milieu de chaîne des MCB pour calculer la taille mémoire disponible.

Vous remarquerez aussi que le PC sur lequel ces tests ont été exécutés semble n'avoir que 652.288 octets de mémoire, c'est-à-dire 3072 octets de moins que les 640K (655.360 octets) du DOS. Il s'agit en fait de mémoire cachée par le DOS. Le PC est équipé d'une carte graphique Paradise CGA. Cette carte nécessite un programme résident appelé MGC.COM (voir fichier AUTOEXEC.BAT dans l'exemple 3). Mais pourquoi MGC.COM n'apparaît-il pas dans l'un des blocs mémoires ?

Ce programme occupe en fait les 3072 octets hauts de la mémoire. Lorsqu'il s'installe, il modifie le bloc mémoire DOS à l'adresse 40H:13H, et effectue une réinitialisation à chaud du système sans test mémoire. La raison de cela est que MGC.COM doit être présent en mémoire avant même que le DOS ne soit chargé (par exemple pour des jeux qui sont lancés automatiquement

```
C>GRAPHICS

C>MCB
=============================================================
MCB MCB   ID PID      MB  PAR- ENV  OWNER
NO. SEG            SIZE ID   BLK?
=============================================================
01  0973 M  0008  16352 022B N    IBMDOS.COM/MSDOS.SYS
02  0D72 M  0D73   3376 0D73 N    COMMAND.COM COPY #1
03  0E46 M  0000     48 0F14 N    FREE MEMORY CONTROL BLOCK
04  0E4A M  0D73    160 0D73 Y    COMMAND.COM COPY #1
05  0E55 M  0E5F    128 0D73 Y    C:\DOS\FASTOPEN.EXE
06  0E5E M  0E5F   2896 0D73 N    C:\DOS\FASTOPEN.EXE
07  0F14 M  0F1E    128 0D73 Y    C:\DOS\GRAPHICS.COM
08  0F1D M  0F1E   2144 0D73 N    C:\DOS\GRAPHICS.COM
09  0FA4 M  0FAE    128 0D73 Y    C:\TURBOC\DEV\MCB.EXE
10  0FAD M  0FAE  71232 0D73 N    C:\TURBOC\DEV\MCB.EXE
11  2112 Z  0000 516816 0F14 N    FREE MEMORY CONTROL BLOCK
=============================================================
```

Exemple 4 : **Exécution de la commande DOS GRAPHICS**

```
C>COMMAND

The IBM Personal Computer DOS
Version 3.30 (C)Copyright International Business Machines Corp 1981,1987
             (C)Copyright Microsoft Corp 1981, 1986

C>MCB

=====================================================================
MCB MCB  ID PID     MB  PAR- ENV  OWNER
NO. SEG            SIZE ID   BLK?
=====================================================================
01  0973 M  0008  16352 022B N    IBMDOS.COM/MSDOS.SYS
02  0D72 M  0D73   3376 0D73 N    COMMAND.COM COPY #1
03  0E46 M  0000     48 0F14 N    FREE MEMORY CONTROL BLOCK
04  0E4A M  0D73    160 0D73 Y    COMMAND.COM COPY #1
05  0E55 M  0E5F    128 0D73 Y    C:\DOS\FASTOPEN.EXE
06  0E5E M  0E5F   2896 0D73 N    C:\DOS\FASTOPEN.EXE
07  0F14 M  0F1E    128 0D73 Y    C:\DOS\GRAPHICS.COM
08  0F1D M  0F1E   2144 0D73 N    C:\DOS\GRAPHICS.COM
09  0FA4 M  0FAD    112 0FAD N    COMMAND.COM COPY #2
10  0FAC M  0FAD   3376 0FAD N    COMMAND.COM COPY #2
11  1080 M  0FAD    160 0FAD Y    COMMAND.COM COPY #2
12  108B M  1095    128 0FAD Y    C:\TURBOC\DEV\MCB.EXE
13  1094 M  1095  71232 0FAD N    C:\TURBOC\DEV\MCB.EXE
14  21F9 Z  0000 513120 0F14 N    FREE MEMORY CONTROL BLOCK
=====================================================================
```

Exemple 5 : **Lancement d'une deuxième instance de COMMAND.COM**

Robert Moore est ingénieur senior consultant à El Segundo, Californie pour le projet de radar des F-14.
On peut le joindre à 2126 Prosser Ave., Los Angeles, CA 90025.

La Gestion Mémoire sous MS - DOS : Copyright **Dr. Dobb's Journal**, M&T Publishing Inc. Redwood City, CA USA 1990. All right reserved.

depuis une disquette). Le système MCB/MB n'est utilisé que par le DOS, et ne convient pas à ce cas précis.

Conclusion

Les applications possibles de ce programme sont multiples. Vous pouvez par exemple examiner les effets des fonctions de gestion de mémoire du DOS en détails. Vous pouvez également imaginer une manière d'effacer un programme résidant de la mémoire, en créant un trou dans la mémoire. Et avec la possibilité de localiser les variables de l'environnement, vous pouvez imaginer un programme qui permettrait l'édition plein écran de la variable PATH par exemple, sans être obligé d'utiliser l'ennuyeuse commande de SET. Et sans aucun doute, vous trouverez d'autres utilisations de ce programme.

```
C>EXIT

C>MCB

=====================================================================
MCB MCB  ID PID     MB  PAR- ENV  OWNER
NO. SEG            SIZE ID   BLK?
=====================================================================
01  0973 M  0008  16352 022B N    IBMDOS.COM/MSDOS.SYS
02  0D72 M  0D73   3376 0D73 N    COMMAND.COM COPY #1
03  0E46 M  0000     48 0F14 N    FREE MEMORY CONTROL BLOCK
04  0E4A M  0D73    160 0D73 Y    COMMAND.COM COPY #1
05  0E55 M  0E5F    128 0D73 Y    C:\DOS\FASTOPEN.EXE
06  0E5E M  0E5F   2896 0D73 N    C:\DOS\FASTOPEN.EXE
07  0F14 M  0F1E    128 0D73 Y    C:\DOS\GRAPHICS.COM
08  0F1D M  0F1E   2144 0D73 N    C:\DOS\GRAPHICS.COM
09  0FA4 M  0FAE    128 0D73 Y    C:\TURBOC\DEV\MCB.EXE
10  0FAD M  0FAE  71232 0D73 N    C:\TURBOC\DEV\MCB.EXE
11  2112 Z  0000 516816 0F14 N    FREE MEMORY CONTROL BLOCK
=====================================================================
```

Exemple 6 : **La deuxième instance de COMMAND.COM reste en mémoire jusqu'à ce qu'une commande Exit soit saisie.**

```
C>CHKDSK

 21309440 bytes total disk space
 53248 bytes in 2 hidden files
 59392 bytes in 28 directories
 9635840 bytes in 580 user files
 11560960 bytes available on disk

 652288 bytes total memory
 588064 bytes free
```

Exemple 7 : **Exécution de la commande DOS CHKDSK pour obtenir l'espace mémoire disponible et le comparer à celle indiquée par MCB.C**

3614
DEBUG
Magazine

PROGRAMMATHEQUE DE DEBUG MAGAZINE

Disquette Listing Debug Magazine n° 1 (juin 1989)

Compression de données par sir Geoffroy Hales en PASCAL.
Liaison BASIC et Assembleur par Alexeï Hoffman, en ASM.
Capture d'écrans par Raoul Saltzberg, en ASM.
Rendez compatible l'incompatible par P. Combe, en PASCAL.

Disquette Listing Debug Magazine n°2 (juillet/août 1989)

Clin d'oeil WINDOWS par Ch. Similia, en C.
BASIC/Assembleur - fenêtres faciles par R. Salzberg, ASM, BASIC.
INT 24h par J.P.Rieux, PASCAL.
8 fois plus vite par Alexeï Hoffman, PASCAL.
Halala! Interface DOS en langage naturel par Chap Humphrey, BASIC
Turbo Professionnal 5.0 par Serge Grun, PASCAL

Disquette Listing Debug Magazine n°3 (septembre / octobre 1989)

Votre PC surveillé par G. Ducamp, BASIC
Création programmes TSR en Turbo Pascal par Ken Pottenbaum, PASCAL.
Techniques d'affichages rapides par Thierry Ruiz, ASM, PASCAL.
Automatiser le désassemblage par G. de Brebisson, PASCAL.
SCRIBE, un petit éditeur de textes par P.Gaithier, ASM.
Mode réel : un peu de magie par P. Gaithier, C.

Disquette Listing Debug Magazine n°4 (novembre / décembre 1989)

Les programmes résidents en Turbo Pascal par Ken Pottenbaum, PASCAL.
SDIR, affichage de répertoires et recherche de fichiers par Alexeï Hoffman, C.
INT2E, l'astuce EXEC par Alexeï Hoffman, ASM.

Disquette Listing Debug Magazine n°5 (janvier / avril 1990)

Quick Pascal en Quick Pascal par J. Mouhanna et M. Vose, PAS.
Vidons les disquettes par G. de Brebisson, PAS.
La 4ème dimension par P. Sagaut, PAS.
BUFFER : tampon imprimante résidant par T. Ruiz, ASM.
DOS et la mémoire étendue par S. Grun, C.

Disquette Listing Debug Magazine n°6 (mai / août 1990)

Assembleur toujours vivant par M. Abrash (DDJ), ASM.
Les astuces du métier par Tim Patterson (DDJ), ASM.
Cohabitation avec l'Assembleur par K. Wright (DDJ), ASM, PASCAL, C.
Programmer sur SDK WINDOWS (1ère partie) par S. Grun, C.
HARDCOPY sur l'imprimante de votre choix par Alexeï Hoffman, ASM.
CAPS - un utilitaire ...inutile ? par G. de Brebisson,PASCAL.
La récupération d'erreurs en TP 4,5 et 5.5 par L. Chambaret, PASCAL.

Disquette Listing Debug Magazine n° 7 (septembre / octobre 1990)

Accéder au matériel en mode protégé 386 par S. Fried (DDJ), C.
Initialisation du mode protégé 386 par N. Margulis (DDJ), ASM.
Débogueur maison style 386 par Al Williams (DDJ), ASM, C.
Modifiez les caractères de votre PC par S. Arbiol, PASCAL, BASIC.
Fenêtres pour C par Alexeï Hoffman, C.

Disquette Listing Debug Magazine n° 8 (novembre 1990)

Cipher par C.E. Burton, C.
Cipher/Encipher par J. L. Dean, C, EXE), un exemple d'implémentation de chiffrage fractionnement.
Private Line version 7.02 par Private Line Entreprises, programme complet de chiffrage au standard D.E.S, shareware.
Un Générateur de sons en mode protégé i 386, par Fried, C.
Crypteur D.E.S par T. Gempp, PASCAL.
Protect et Vérif par A. Hoffman, ASM, COM.
Outil de chiffrement à clé cachéee, par L.Michener, C.

Disquette Listing Debug Magazine n° 9 (janvier 1991)

Fractal , par P.Morel Fourrier, C, EXE.
Minitel, émulateur de minitel, par D. Chassignet, ASM, C, EXE.
SPA, par Craig A. Lindley, PASCAL, ARC.
Kermit, Kermit en Modula 2, par Brian R. Anderson.
Cheapcom, logiciel de communication pour Mac, par Don Gaspar.
Fractal Graphics, Shareware par Dick Oliver, Cedar Software, C.

Disquette Listing Debug Magazine n° 10 (février 1991)

Kermit pour OS/2 par Brian R. Anderson.
Protect & Vérif par Bruno Vasseur, ASM.

Disquette Listing Debug Magazine n° 11 (mars 1991)

Kermit pour OS/2 par Brian R. Anderson
Protect & Vérif (nouvelle version) par Bruno Vasseur ASM.
GAG.

Disquette Listing Utilitaires

LHARC

de Haruyasu Hoshizaki
PC SIG # 2018

Pour commander ces disquettes, utiliser le coupon détachable, encarté.

Un gestionnaire de mémoire cache

Comment éviter la saturation des disques en utilisant une mémoire cache.

par Alan Deikman
Traduction par Pierre Morel-Fourrier

Avec la plupart des systèmes d'exploitation, les applications effectuent des requêtes de lecture et écriture qui paraissent s'exécuter instantanément - le débit dépend principalement de la vitesse d'entrée sortie du périphérique utilisé. Cette contrainte est supportable pour certains périphériques. Après tout, il n'y a généralement pas grand chose à faire pendant qu'un programme attend une saisie au clavier.

Toutefois, pour les accès aléatoires sur disque, les données recherchées qui peuvent n'arriver qu'après un délai de 80 millisecondes, peuvent avoir déjà été utilisées quelques instants auparavant. Si un programme a la prévoyance et l'intelligence de conserver ce bloc de données en mémoire, il peut n'y avoir aucun accès disque du tout, et donc pas de délai de 80 millisecondes.

Lors des accès disque en général, certains blocs de données sont lus, relus encore et encore. D'autres blocs de données ne seront consultés qu'épisodiquement. La partie système d'un disque MS-DOS est habituellement la partie lue la plus souvent. Elle contient la liste des fichiers et la table d'allocation des fichiers (FAT).

Dans un cas idéal, un système d'exploitation multitâche passe le contrôle à d'autres tâche quand l'une d'elle effectue un accès disque. Le contrôle est rendu à la tâche quand les données ont été lues sur disque. Le débit global du système est ainsi avantagé au détriment d'une tâche particulière. Et même, il n'y a aucun avantage à permettre à une tâche de forcer le système d'exploitation à effectuer plus d'accès disque que nécessaire.

Si de tels programmes sont fréquemment utilisés, la capacité du système peut être grandement améliorée si chaque programme est optimisé afin de réduire les accès disque. Pour des applications effectuant beaucoup d'opérations d'entrée sortie, ce détail devient déterminant. Un programme de comptabilité ou une base de données n'effectuent pas beaucoup de calculs. La plupart du temps d'exécution (après la saisie des données) est passé à rechercher ou à stocker des informations.

Dans ce contexte, il existe deux manières d'améliorer la performance :
1. Acheter un matériel plus puissant.
2. Réduire le nombre d'accès disque nécessaires.

Le cache disque représente l'un des moyens les moins chers pour améliorer les performances d'un programme.

La théorie du cache

Le cache disque est basé sur une zone de mémoire réservée qui est utilisée pour stocker les blocs de données les plus utilisés. Chaque bloc est identifié par un numéro de bloc. Lorsqu'une requête de lecture de bloc survient, la mémoire cache est testée en premier pour vérifier si le bloc demandé s'y trouve ou non. S'il s'y trouve, aucun accès disque n'est effectué, et le programme appelant utilise une copie du bloc de données. Ce bloc est marqué « dernier bloc utilisé » (Most Recently Used, MRU).

S'il se trouve qu'un accès disque est nécessaire (car le cache ne contient pas le bloc demandé), le bloc de données est obtenu par une lecture sur disque habituelle. Toutefois, avant que le bloc ne soit utilisé, il est recopié dans la zone de cache, à la place du bloc qui était le moins utilisé (Least Recently Used, LRU) si celui-ci existe. Ce nouveau bloc est marqué MRU.

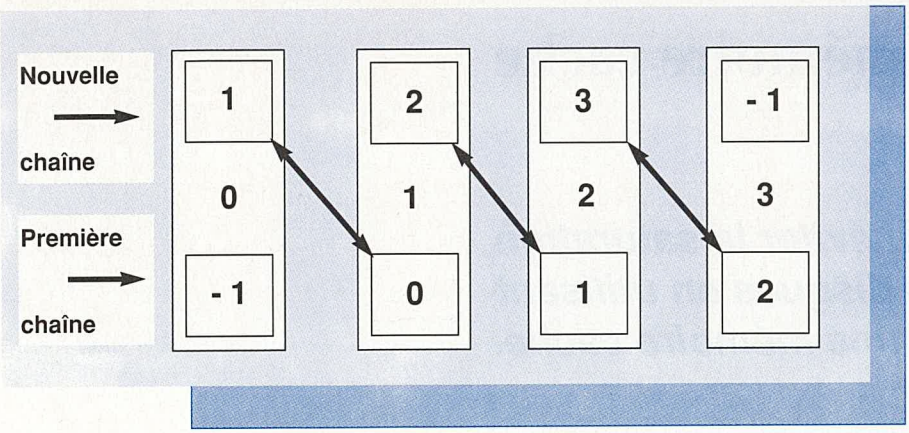


Figure 1 : **Chaîne LRU/MRU initiale**

Quand un bloc de données est écrit sur disque, il est également recopié dans le cache et marqué MRU. Dans la plupart des applications, il est préférable d'écrire immédiatement ce bloc de données sur disque. Mais il est possible aussi de ne l'écrire sur disque que quand ce bloc va devenir LRU et jeté du cache. En effet, le bloc modifié peut ne pas avoir besoin d'être écrit immédiatement sur disque.

Les routines de cet article utilisent une double liste chaînée, appelée chaîne LRU/MRU. La fonction cacallo() initialise la liste chaînée pour y inclure tous les blocs du cache, comme indiqué figure 1. Une valeur -1 indique la fin de la chaîne.

Quand un bloc est marqué MRU, il est extrait de la liste et réinséré à la fin. Cette opération est effectuée par la fonction cacnew(). Lorsque, par exemple, le bloc 2 de la chaîne initiale est marqué MRU, la chaîne devient comme indiqué à la figure 2.

La taille du cache

Il y a un compromis à trouver entre la taille de la mémoire cache et son efficacité. Si le cache est trop petit, la probabilité de retrouver un bloc dans le cache est faible. D'un autre côté, si le cache est trop grand, trop de temps CPU est utilisé à parcourir et mettre à jour la liste des blocs dans le cache. Le manuel MS-DOS met en garde contre ce problème à propos de la commande BUFFERS de CONFIG.SYS. La commande BUFFERS crée un cache avec des enregistrements de 128 octets.

La courbe de la figure 3 représente l'efficacité d'un cache effectuant des accès aléatoires en fonction de sa taille. L'extrémité gauche de la courbe représente un cache de taille nulle. Pour des petites tailles de mémoire cache, la performance diminue. Elle augmente ensuite au fur et à mesure que la taille du cache augmente vers un optimum théorique et diminue ensuite. Le cache devient alors un fardeau plus qu'une aide.

En pratique, les accès disque ne sont pas aléatoires. Il existe même des cas pour lesquels un cache n'est pas utile. Considérons par exemple le cas d'un fichier lu du début à la fin. Aucun bloc n'est lu deux fois, et l'utilisation d'un cache ralentira le système. Si le fichier contient un bloc de plus que la taille du cache, et que ce fichier doit être lu plusieurs fois du début à la fin, aucun accès à la mémoire cache ne sera un succès. En fait, il n'est pas désirable d'appliquer systématiquement un cache disque pour la lecture de fichiers séquentiels de taille inconnue.

Dans le cas où le fichier est plus petit que la taille du cache, un cache est superflu car il est alors préférable de charger complètement le fichier en mémoire et d'accéder directement aux blocs du fichier, sans utiliser de liste chaînée de blocs MRU/LRU.

L'efficacité d'un cache dépend de nombreux facteurs. La fonction cacstat() affiche les statistiques de succès du cache pour pouvoir en ajuster la taille et obtenir ainsi les performances optimales. Les valeurs indiquées sont le nombre de succès, d'échecs, et d'ajouts de blocs dans le cache. La somme du nombre de succès et d'échecs représente le nombre total d'accès au cache. Le taux de succès optimum dépend de nombreux facteurs comme par exemple le rapport entre les temps d'accès de la mémoire et du disque.

Les caches multiples

L'un des défauts les plus visibles d'un système d'exploitation qui utiliserait des caches pour toutes ses entrées sorties disque est que le système d'exploitation serait incapable de distinguer les données qu'il faut conserver dans le cache et celles qu'il ne faut pas. De plus, la plupart des systèmes ne peuvent allouer de mémoire cache dynamiquement. Il en résulte que tous les blocs utiles du cache du système peuvent être jetés, juste parce qu'une application a effectué une lecture d'un fichier, en remplissant le cache d'enregistrements qui ne seront jamais lus une deuxième fois. Si le système est multitâche ou multiprocesseur, la situation est encore pire puisque d'autres programmes peuvent envahir l'espace cache.

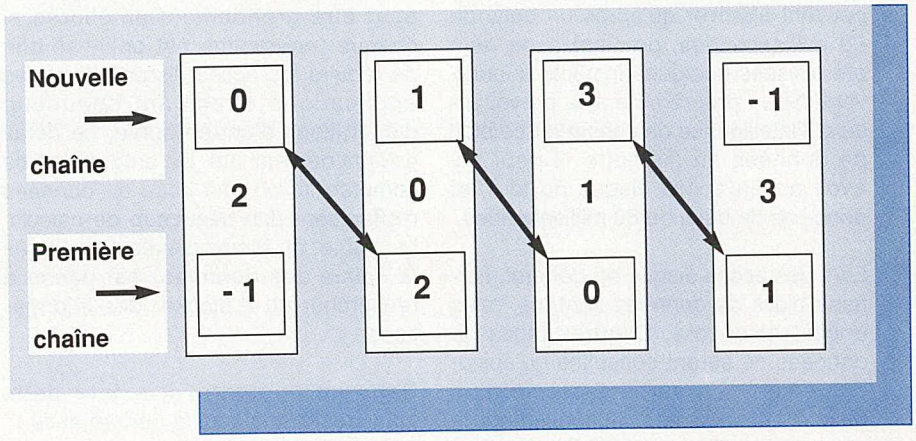


Figure 2 : **Après que le bloc 2 soit devenu MRU**

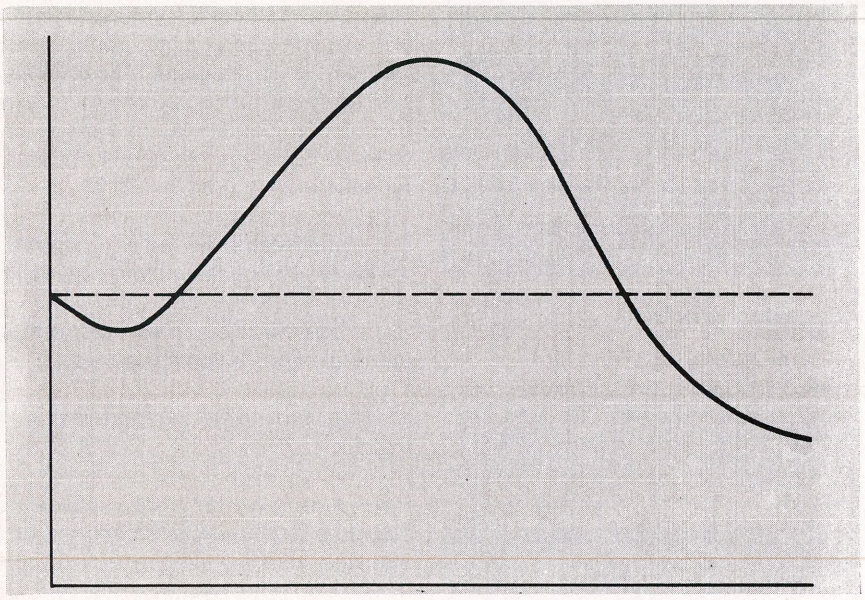


Figure 3 : **Taille du cache et performance**

Le programmeur d'une application sait en général quels sont les fichiers qui doivent être cachés et ceux qui ne doivent pas l'être. Un bon exemple d'application de cache disque est le stockage des tables temporaires lors d'une compilation. Un fichier est utilisé lorsqu'il n'y a plus de mémoire disponible. Un cache est aussi particulièrement efficace lors de la recherche dans des arbres de type B-arbre.

Pour cela, toutes les routines de cet article utilisent une structure sur laquelle pointe une variable. Ce pointeur est transmis par le programme appelant. Il pointe sur une zone mémoire servant de cache disque. Ce procédé permet d'utiliser simultanément un nombre quelconque de caches de tailles différentes.

La fonction cacallo() utilise la fonction standard malloc() pour allouer la mémoire du cache. Elle retourne un pointeur sur la structure racine du cache. Ce pointeur sera ensuite utilisé pour identifier le cache lors des appels des autres fonctions de gestion du cache. Tous les paramètres du cache, ainsi que les pointeurs vers les autres objets sont contenus dans cette structure de type CACDS.

Quatre paramètres sont indispensables pour définir un cache:

- le nombre d'enregistrements dans le cache
- la longueur de chaque enregistrement
- un pointeur sur une fonction externe qui traite la libération des enregistrements
- un mot long d'identification du cache à transmettre à la fonction externe

Si le cache ne doit pas différer l'écriture des enregistrements sur disque, la fonction externe de traitement de libération des enregistrements n'est pas nécessaire. Dans ce cas, la valeur du pointeur sur la fonction doit être nulle.

Le mot d'identification est utilisé lorsque la routine externe doit gérer la libération des blocs de plusieurs caches. Ce mot permet à la routine externe d'identifier le cache auquel appartient l'enregistrement qui lui est transmis.

L'utilisation des routines de cache

Toutes les routines et la déclaration de la structure de cache sont dans le fichier cache.h. Il n'est pas nécessaire pour le programme appelant de connaître cette structure car il n'accède jamais directement aux champs de cette structure CACDS.

Cache.h déclare cette structure avec une instruction typedef, et ainsi, le programme appelant peut respecter la norme ANSI en déclarant une variable externe ou locale de type CACDS. Certains compilateurs acceptent toutefois de stocker la valeur retournée de cacallo() dans un pointeur caractère.

Le fichier cache.h fournit aussi les déclarations de fonctions nécessaires aux contrôles de types du compilateur. Le listing 2 liste les routines de gestion du cache. Elles peuvent être compilées séparément et/ou utilisées comme une librairie. Le listing 3 est un simple exemple d'utilisation de ces routines.

Après que le cache ait été initialisé, le programme doit tester que le bloc demandé est bien présent. Pour cela, la fonction cacfind() doit être appelée. Si l'enregistrement du cache dont le numéro est indiqué par la variable num n'est pas trouvé, un pointeur nul est retourné, et le programme appelant peut alors traiter ce cas. En général, il s'agit alors d'effectuer un read() pour lire le bloc demandé, puis d'ajouter ce bloc dans le cache.

Si l'enregistrement est trouvé par cacfind(), un pointeur sur l'enregistrement spécifié est retourné par la fonction au programme appelant. Cet enregistrement est supprimé de la chaîne LRU, et inséré à la fin de la chaîne MRU. Par exemple, supposons que le programme appelant utilise des enregistrements de 512 octets, et qu'il désire lire l'enregistrement dont le numéro est stocké dans la variable recn. Le pointeur externe cache est initialisé avec la valeur retournée par un appel à la fonction cacallo(). Le code correspondant à cette recherche (sans traitement d'erreurs) figure dans l'exemple 1.

Cet exemple n'ajoute pas les enregistrements lus sur disque dans le cache. Un enregistrement lu par la fonction read() devrait être ajouté dans le cache à la fin de la liste MRU car, en fait, c'est le dernier enregistrement qui a été lu.

Chaque fois qu'un enregistrement doit être ajouté dans le cache, la fonction cacnum() est appelée. Il est important de noter que la routine cacnum() ne teste pas si l'enregistrement est déjà présent dans le cache. Chaque appel à la fonction cacnum() doit donc être précédé par un appel à la fonction cacfind().

La fonction cacnum() effectue le travail suivant :
1. L'enregistrement qui a été le moins utilisé (LRU) dans le cache est recherché.
2. Si ce bloc a été marqué modifié, la routine externe de libération de bloc écrit le bloc sur disque.
3. Ce bloc devient le bloc le plus récemment utilisé (MRU).

```
char buffer [512];
long recn;
char *rec;
{
if ((rec = cacfind (cache, recn) ==NULL) {
    /* L'enregistrement n'est pas dans le cache, il doit être lu sur disque */
    lseek (fd, recn* 512, 0);
    read (fd, buffer, 512);
    rec = buffer;
/* rec pointe sur l'enregistrement à traiter */
}
```

Exemple 1 : **Lecture de blocs de 512 octets**

```
char buffer [512];
long recn; /* No du l'enregistrement à écrire */
char *rec; /* Pointeur vers le bloc dans le cache */

{
if ((rec = cacfind (cache, recn) ==NULL)rec = cacnum (cache, recn);
memcpy (rec, buffer, 512); /* Recopie les données dans le cache */
lseek (fd, 512 * recn, 0);
write (fd, rec, 512);
}
```

Exemple 2 : **Écriture de blocs de 512 octets**

4. Le numéro de bloc passé en paramètre est appliqué à cet enregistrement.
5. Un pointeur sur ce bloc est retourné.

Pour éviter de recopier les blocs en mémoire, et d'avoir à allouer de la mémoire, le programme appelant accède aux données par l'intermédiaire des pointeurs retournés par les fonctions cacfind() et cacnum().

Pour terminer cet exemple de lecture d'enregistrement, le code suivant (sans gestion d'erreur) peut être utilisé :

```
long recn;
char *rec;
{
if ((rec = cacfind(cache, recn)) == NULL)
{
rec = cacnum(cache, recn);
lseek(fd, recn*512, 0);
 read(fd, rec, 512);
}
```

L'écriture des données représente un processus similaire. Un enregistrement qui est écrit dans le cache est aussi marqué MRU via les fonctions cacfind() et cacnum(). Si l'enregistrement est déjà dans le cache, il n'y a pas besoin d'appeler la fonction cacnum(). La routine complémentaire à la routine de lecture de bloc est représentée par l'exemple 2.

La gestion des enregistrements libérés

Comme mentionné plus haut, il est quelquefois avantageux d'ajouter les enregistrements modifiés dans le cache sans les enregistrer immédiatement sur disque. L'opération d'écriture ne sera effectuée que quand ce bloc modifié deviendra LRU et qu'il devra être jeté du cache par manque de place. Ce genre de processus peut typiquement être utilisé pour les fichiers temporaires (comme par exemple une table de symboles) lorsque le fichier en question n'a pas besoin d'être créé à moins que les données ne débordent du cache. Mais dans la plupart des cas, il vaut mieux écrire immédiatement les enregistrements sur disque, comme indiqué dans l'exemple 2.

Si l'écriture des enregistrements doit être différée, les troisième et quatrième paramètre de la fonction cacallo() sont utilisés, et la fonction cacproc() est utilisée pour marquer les enregistrements à écrire sur disque avant que l'espace mémoire qu'ils occupent ne soit écrasé par un autre enregistrement. Cette fonction de libération fournie par le programme appelant contient trois paramètres :

1. L'identificateur du cache (le quatrième paramètre de la fonction cacallo())

2. Un identificateur d'enregistrement (généralement le numéro du bloc)

3. Un pointeur vers l'enregistrement à traiter

Pour implémenter cela, la routine d'écriture d'un enregistrement doit être modifiée, comme indiqué à l'exemple 3.

Le programme appelant doit fournir à la fonction cacallo() un pointeur sur une fonction qui écrit un enregistrement sur disque. Quand un bloc marqué modifié doit être libéré, cette fonction est appelée. Une application de ceci est présentée dans l'exemple 4.

Dans certains cas, un bloc marqué modifié devient obsolète. La fonction cacupr() est alors appelée. Cette routine empêchera un bloc d'être envoyé à la fonction externe d'écriture d'un bloc sur disque.

La fermeture du cache

La fonction cacallo() fait appel à malloc() pour allouer la mémoire du cache. On libère cette zone mémoire en appelant la fonction cacfree(). Tous les enregistrements modifiés et non écrits sur disque sont traités en appelant la fonction cacflsh(), puis la mémoire est libérée par un appel de la fonction free().

La taille optimale du cache

Les routines de gestion de cache disque présentées dans cet article conviennent mieux à de petites et moyennes tailles de cache, et à des tailles d'enregistrement fixes et assez grandes. Typiquement, les« enregistrements » cachés sont des blocs de données, qui contiennent de plus petits enregistrements. Quand les enregistrements sont petits, une recherche linéaire devient rapidement inefficace. La figure 4 représente un diagramme où des programmes d'interface gèrent des requêtes provenant d'applications.

```
char buffer [512];
long recn; /* No du l'enregistrement à écrire */
char *rec; /* Pointeur vers le bloc dans le cache */
{
if ((rec = cacfind (cache, recn) ==NULL)rec = cacnum (cache, recn);
memcpy (rec, buffer, 512);          /* Recopie les données dans le cache */
cacproc (cache, recn);          /* Marque le bloc à traiter */
}
```

Exemple 3 : **Écriture différée sur disque**

Alan Deikman,
Software Services,
Menlo Park,
CA 94026-2106.
Alan est consultant management, marketing et informatique.
Il est consultant depuis 1978, et indépendant depuis 1985.

```
write_cache (idnt, recn, recb)
long     idnt;    /* Identificateur du cache */
long     recn;    /* Numéro d'enregistrement */
char     *recb;   /* Tampon de l'enregistrement */
{
      lseek (fd, recn * 512, 0);
      write (fd, recb, 512);
      return;
}
```

Un gestionnaire de mémoire :
copyright **Dr Dobb's Journal**
M&T Publishing,
Inc, Redwood City,
CA USA 1990.
All right reserved.

Exemple 4 : **Exemple de fonction d'écriture d'un bloc**

Conclusion

Les routines présentées ici peuvent être appliquées à tout type d'application. Elles ont été originalement écrites sous Unix System III, un système à base de 68000, et ont été portées sous d'autres environnements. Ces routines fonctionnent très bien sous MS-DOS dans tous les modèles mémoire du compilateur Microsoft-C, bien qu'un cache de plus de 63 Ko ne puisse fonctionner.

PARLONS - EN

POLEMIQUE

La notion de partialité

par Olivier Dahan

Suite à la première partie de l'article de Mr. Bourne (OUF, Debug Magazine n°10), je tiens à apporter des observations suivantes:

Un « shareware » est bien un produit commercial, contre toute attente et à l'inverse de ce qui est dit .... D'ailleurs, s'il n'existe qu'une alternative pour une porte (être fermée ou ouverte), il n'y en a qu'une pour un produit (quel qu'il soit): être gratuit ou payent. Pour le logiciel, gratuit se dit « freeware » ou domaine public. Payant se traduit sous deux formes : l'une habituelle l'autre - shareware. Il ne suffit d'ailleurs pas de traduire le terme par partagiciel pour traduire les mentalités et implanter le concept en France où il ne reste, dans les esprits, qu'une variante du freeware.

Un shareware est un produit bénéficiant d'un support légal tout à fait clair : l'auteur ne souhaitant pas dépenser son argent en pub ou ne voulant pas dépendre d'un éditeur, diffuse une version de son produit sur les BBS (messageries électroniques, nombreuses aux USA). Vous disposez normalement de 30 jours pour l'essayer, au delà, vous êtes tenus de payer la licence! La notion de choix n'existe pas et il s'agit d'un acte de vente légal, cerné par toutes les règles juridiques délimitant un tel acte.

La seule différence notable avec un produit commercial « classique » est donc la possibilité d'essayer avant d'acheter. C'est la notion du prêt, tout simplement qui, d'ailleurs, est de plus en plus utilisée par le circuit commercial classique.

Notons, pour conclure, que le shareware n'est pas un « demi-freeware » mais bien un produit sous copyright (donc protégé par les lois sur le droit d'auteur), ne pouvant être utilisé que par autorisation de l'auteur (moyennant finance en général). D'ailleurs, le procès entre les concepteurs de ARC et de PKZIP (ex PKARC), deux produit shareware, prouve qu'il s'agit bien d'un business, avec profits, copyrights, procès et batailles purement commerciales ...

La confusion est née de la teneur, au début du shareware, des messages des auteurs: « SI CE PROGRAMME VOUS PLAîT, ENVOYEZ- MOI $5 » Certains y ont vu la possibilité de ne pas payer, d'ou une certaine confusion avec le freeware. L'ambiguïté de ce type de message a disparu des shareware récents et aurait du être balayé par la raison: car si le logiciel ne plaît pas, vous n'êtes pas sensé l'utiliser!... On retrouve ainsi toutes les caractéristiques d'un produit commerciale classique, seul le réseau de distribution change. Ainsi, le shareware apparaît être plus un type de vente qu'un type de produit (à la différence du freeware), c'est un système de diffusion, comme la VPC, le marketing direct et autres techniques.

La cohérence

Soyons cohérents, justement.

Le shareware, en tant que mode de diffusion économique d'un produit, ne propose pas aux utilisateurs le même suivi, la même qualité de prestation que les produits diffusés autrement. Certes, de grands noms du shareware sont aujourd'hui richissimes et proposent Hot Line (Ligne Chaude - NDLR), serveurs, etc., mais il n'en restent pas moins attachés aux principes de base. Ainsi, pour un produit classique, c'est normalement l'éditeur qui vous tient averti des nouvelles versions (ce qui est indispensable dans le cadre d'une utilisation« pro » d'un produit). Pour les shareware, c'est à l'utilisateur de se connecter sur les serveurs qu'il connaît (ou celui du concepteur) afin de voir, régulièrement, si quelque chose de nouveau a été fait. Charge à lui de « télécharger » les fichiers qui l'intéressent. Aucune mise à jour automatique par disquette, aucun courrier régulier ne lit le concepteur et l'utilisateur (toujours en précisant que les choses changent puisque ceux qui ont fait fortune avec le shareware proposent souvent des versions « commerciales » plus classiques avec les services que cela sous-entend).

Illustration de cet état de fait, en concordance avec les produits utilisés par l'auteur de l'article, le système SCAN et CLEAN-UP, l'antivirus shareware de McAFee. Selon les copies d'écran de l'article, Mr Bourne utilise la version 62 de SCAN. Il faut savoir que la version 72 actuelle reconnaît plus de cent virus supplémentaire ... S'il se croie protégé c'est une erreur! Par contre, pour obtenir les toutes dernières versions, il faut se connecter sur le serveur de McAFee en Californie (problèmes de lignes, coût des communications, etc.)... Jamais McAFee ne lui enverra une note l'avertissant que son produit est dépassé et qu'il est plus qu'obsolète face au nombre de croissant de virus sur le marche. McAFee développe d'ailleurs un autre produit, PRO-SCAN (Professionnal Scan), vendu normalement et dont l'auteur de ces lignes est le représentant exclusif pour la France (on retrouve alors la notion de suivi du client, d'abonnements aux mises à jour, de lettres d'informations, en dehors des caractéristiques techniques du produit qui sont différentes, bien entendu).

Premier problème de cohérence, donc.

Le shareware est né de la multiplicité des serveurs de fichiers type BBS aux USA (et de l'équipement massif des utilisateurs en modems puissants). En France il n'existe rien, sauf quelques rares serveurs (type Ouf, Suptel, mais ils sont soumis à abonnement en général ce qui s'écarte énormément du concept américain), ainsi les sources d'approvisionnement sont rares. Difficulté doublée par le fait que les auteurs américains ne déposent pas leur création ailleurs que sur des serveurs américains... Rappelons aussi qu'aux USA on trouve des modem

[SERVICE LECTEURS : INTERESSANT : 45 / MOYEN :46 / MAUVAIS : 47]

4800 Bds très corrects aux environs de $75 (400F.) alors qu'en France un 2400 Bds se paye encore plusieurs milliers de francs.... L'évolution du shareware est intimement lié au parc de serveurs et à l'équipement des utilisateurs en moyens de communication puissants (Modem supportant les protocoles NMP4 / NMP5 en 9600 Bds pour la plupart des serveurs US aujourd'hui). En France il ne peut exister véritablement de shareware en raison même de l'indigence de notre infrastructure à ce niveau (serveurs, modems, etc.). Ici, hors de Transpac et du Minitel, point de salut! Quand à aller télécharger des produit en 3615, mieux vaut se connecter aux serveurs américains: c'est le même prix et les versions récupérées ne datent pas d'il y a deux ans!

Les produits présentés

Après cette introduction, on comprendra mieux les lacunes et incohérences que je vais ici relever. S'il n'y a rien à dire sur le pilote ANSI.COM (bien qu'il existe des versions spéciales EGA comme ZANSI.COM ce qui serait plus approprié aux besoins de l'auteur de l'article), ni sur SCAN et CLEAN-UP (voir plus haut), il nous faut relever un problème grave concernant le second produit présente: OPTUNE.

OPTUNE est un logiciel commercial depuis le début! Il ne s'agit pas d'un shareware! La version que l'auteur possède est soit marquée « BETA RELEASE » ou bien c'est du piratage simple et pur. En effet, la société Gazelle Systems (42 North University avenue, Suite 10, Provo, Utah, 84601) avait, il y a déjà un ou deux ans, mis son version bêta sur quelques serveurs américains. Elle est arrivée en France (c'est d'ailleurs comme ça que je l'ai connue) et est effectivement utilisée comme un produit fini par certains. En dehors de la violation des droits d'auteur (délicate ici puisqu'il s'agit d'une version bêta...), ce produit n'est pas « stabilisé », selon l'expression d'usage - ce n'est qu'un prototype. Certes, elle fonctionne bien, mais cela n'a rien à voir avec la version commerciale qui propose en plus un système de type « mirror » de PCTOOLS, un formateur de disquettes résidant, qui, bien entendu, fonctionne sur les DOS 4.x et intègre les derniers contrôleurs de disques dur afin d'éviter la destruction de données sur certaines configurations (méfiance donc si vous utilisez la version bêta!).

Un point important à relever. Une incohérence de plus.

Dernière chose, OPTUNE est titré « gestionnaire de disque dur », c'est en appellation non contrôlée... OPTUNE, comme son nom le laisse entendre, est tout simplement un optimiseur (défragmentation des fichiers, optimisation de l'interlacement non destructive, tri des répertoires, contrôle de qualité des disques, etc.). Il ne gère rien. PCTOOLS version 6, par exemple, est un gestionnaire de disque dur (si ce terme à un sens d'ailleurs, car il peut être confondu avec sa traduction américaine Hard Disk Driver, ce qui n'a absolument rien à voir...). S'il n'y a rien à relever non plus pour QEDIT (bien que ce ne soit pas le meilleur du genre, mais cela est subjectif), ni sur LIFEGUARD, il est important de soulever deux problèmes: encore la cohérence et ensuite la protection antivirale. **La cohérence.** Tant annoncée et tant égratignée... Rappelons qu'un produit comme PCTOOLS V6 FR (je ne touche rien sur ce produit, donc pub gratuite ...) intègre un optimiseur de disque dur (moins rapide que OPTUNE mais bon), un éditeur de texte plus sympathique et plus ergonomique que QEDIT, un système de protection des données plus complet que LIFEGUARD, etc... Sans parler du support souris, de la francisation, de l'assistance en France et pas à l'autre bout du monde... PCTOOLS ne coûte pas cher (avec tout ce qu'il intègre: cache disque, Desktop résidant, DiskFix, etc.) et représente une solution bien plus cohérente que l'emploi de petites bidouilles séparées. Donc pas de cohérence dans ce qui est présenté dans l'article, mais au contraire un chaos infernal entre faux-freeware bêta test (Optune), versions obsolètes (Scan), et surtout, disparité des outils non traduits là où un seul bon produit ferait tout, mieux, et légalement.... **La protection antivirale.** Déjà abordée avec SCAN (voir plus haut) concernant les risques énormes qu'il y à se croire protéger par SCAN62 en février 91, nous allons continuer (et finir) avec les autres produits présentés par Mr Bourne (en attendant la suite de la liste dans le prochain numéro, ce qui justifiera d'autre mises au point).

NOVIRUS

Il est censé assurer la « vérification de l'intégralité des fichier de commande du DOS ». Il s'agit d'un balayeur de fichiers qui, comme cela est dit, range dans un fichier les caractéristiques d'une liste de programmes (ou fichiers) pour ensuite comparer ces informations avec celles qui sont lues lors d'un contrôle de ces même fichiers.**Danger!** Depuis plusieurs mois sont apparus sur le marché des virus appelés par les américains « stealth » certainement pas analogie avec le fameux bombardier F117 (stealth = furtif). Ces virus modernes se caractérise par l'utilisation de techniques sophistiquées leur permettant d'être « furtifs », soit, indétectables. Notons, parmi les plus connus à ce jour, 4096 (Frodo), 512, EDV, The Fish, Joshi, V2000, V2100, V800 et The Whale, pour environ une 15éne de formes virales à peu prés. **Furtif ?**L'un des points majeurs de la technique stealth consiste justement à détourner les appels BIOS, Dos afin de tromper toute vérification du type NOVIRUS. Un virus stealth renvoi toujours une image non infectée d'un programme touché! Même un dump hexadécimal vous donnera une image correcte... Le virus effectue tout simplement une « sauvegarde » des parties modifiées du programme et les renvoie en place et lieu des véritables secteurs qui contiennent l'exécutable... **Diabolique, isn't ?** Ainsi, NOVIRUS est totalement inopérant sur ce type de virus. Le tout contrôlé par SCAN62 auquel il manque une bonne centaine de virus... A titre indicatif (là c'est de la pub!) PROSCAN V2.01b assure, bien entendu, la détection de ce type de virus (notamment par un balayage de la mémoire central afin de désactiver la partie résidante du virus). Nous retiendrons ici que NOVIRUS est totalement obsolète, comme tous les produits effectuant le même type de contrôle antivirale. En fait, cela fonctionne bien pour un système de protection (qui ne peut être parfait, certes), cela est rédhibitoire... Notons, pour conclure, que VACINE, malgré l'emploi de techniques légèrement différentes, est lui aussi une passoire....

NDLR : certaines thèses de cet article ne reflètent pas forcément l'opinion de la rédaction. Nous attendons vos opinions à cet intéressant sujet. D'autre part, nous n'avons jamais pris au sérieux les virus jusqu'il y a un mois ou nous avons trouvé un beau spécimen de Jérusalem sur une disquette tout à fait anodine (origine commercial!). D'autre part, la regrettable mésaventure qui vient d'ariver à notre confrère Soft&Micro (diffusion de virus FRODO sur une disquette encartée dans le N° 74 daté du mai 1991) prouve que le problème n'est pas anodin.

PROTECT ET VERIF

par Jean Baptiste Courtois

A la suite de l'article de Alexeï Hoffman paru dans lenuméro 8, j'ai voulu essayer le système de plombage qu'il y proposait, et j' ai connu les mêmes déboires que Bruno Vasseur (voir le n° 10).

Programme VERIF

- dans la proc _TEST, 9 lignes après l'étiquette RETRY lire JNE NOT_ORIGINAL (et non JNC NOT_ORIGINAL)

- le test de la chaîne écrite sur le secteur 112 de la piste 39 ne peut pas fonctionner correctement puisque les chaînes sur lesquelles portent la comparaison sont différentes, et qu'en plus, la comparaison telle qu'elle est écrite n'opère que sur le 8ème caractère. Il faudrait à mon sens remplacer la valeur de la constante TEST_MSG par 'ATTENTIO', et modifier les deux lignes qui précèdent l'étiquette ORIGINAL comme suit :

```
CLD
REPE CMPSB
JNE  NOT_ORIGINAL
```

Je me suis rendu compte que le listing comportait quelques omissions, que je signale ci-après aux personnes intéressées:

Programme PROTECT

- dans la proc PROTECT, après l'étiquette FIN,
  lire MOV AH,4Ch (ou MOV AX,4C00h)
- dans la proc CHANGE_BASE, au début,
  lire MOV AH,35h
- 4 lignes plus loin,
  lire POP DS (et non PUSH DS)
- enfin, aucune valeur n'est jamais affectée aux variables BASE_SEG et BASE_OFF. Il faudrait insérer dans CHANGE_BASE, à la sortie de l'IT 21h (fonction 35h), les lignes suivantes :

```
MOV AX,ES
MOV Base_Seg,AX
MOV Base_Off,BX
```

Améliorations

J'ai eu l'idée de compléter cet intéressant programme en permettant la protection de disquettes 3"1/2 720 K et en y ajoutant une routine qui déclare les secteurs formatés comme secteurs défectueux, ainsi que le suggérait A. Hoffman dans le but d'éviter des problèmes d'écriture sur ces secteurs. J'ai donc réuni les deux modules en un seul et les ai intégrés dans un programme unique qui propose :

- le formatage de protection et, éventuellement, le marquage des secteurs en question.

- la vérification d'une disquette formatée avec PROTECT.

Ce programme ne fonctionne qu'avec des disquettes 360K et 720K. Il refusera de traiter tout autre type de support (ainsi qu'un disque dur, bien entendu).

J'ignore en effet, quelles tables de paramètres sont utilisées par le DOS

[SERVICE LECTEURS : INTERESSANT = 15 / MOYEN = 16 / MAUVAIS =17]

pour le formatage de disquettes haute densité. Le segment BIOS contient pour sa part quatre DTP différentes (une pour chaque type de disquette), dont les adresses sont fournies par la fonction 18h de l'interruption 13h (Bios AT), mais je ne suis pas sûr que ce soient les bonnes. Les disquettes 720K utilisent quant à elles, les mêmes paramètres que les disquettes 360K.

IBM semblant déconseiller de modifier directement la table des paramètres disque (DTP), le programme modifie seulement à l'aide de l'IT 21h fonction 25h l'adresse marquée en 0000:0078 (IT 1Eh). Le reste des routines PROTECT et VERIF est tout à fait identique à celles parues dans le n° 8, exception faite des détails précisés ci-dessus.

J'y ai seulement ajouté le paramétrage des données dépendantes du type de disquettes. Chacune des deux fonctions assembleur renvoie au programme C une valeur signalant l'échec ou le succès de l'opération.

Déroulement du programme

Sur les machines pourvues de deux lecteurs, il faut préciser sur la ligne de commande, lequel est concerné.

Le programme commence par proposer le choix entre protection et vérification. Si l'utilisateur choisit l'option « protection », une confirmation lui sera demandée. En outre, à chaque entrée au clavier, la touche « Escape » permet de sortir du programme.

Structure de la disquette

Quelle que soit l'option choisie, on vérifie tout d'abord le type de la disquette et ses caractéristiques. Celles-ci sont inscrites dans le premier secteur (secteur de boot) qu'on lit donc (grâce à la fonction C absread() - lecture absolue sur disque). On y trouve, entre autres, les informations de la Table 1.

La fonction type_disk() remplit la structure sdisk avec ces informations, qui nous seront nécessaires par la suite. Il est ensuite procédé au formatage non standard des deux dernières pistes (38 et 39 pour les disquettes 360K, 78 et 79 pour celles de 720K) par la procédure protect(). A l'issue de celui-ci, on propose le marquage des secteurs concernés, consistant à inscrire la valeur FF7h (secteur défectueux) dans les entrées de la FAT qui correspondent aux secteurs en question. Si cette option est retenue, le programme appelle la fonction process_mark().

Offset 11 :	nombre d'octets par secteur	(mot)
Offset 13 :	nombre de secteurs par grappe	(octet)
Offset 14 :	nombre de secteurs réservés	(boot) (mot)
Offset 16 :	nombre de FAT	(octet)
Offset 17 :	nombre d'entrées dans la racine	(mot)
Offset 19 :	nombre total de secteurs	(mot)
Offset 21 :	code de description du support (0FDh = 360K, 0F9h = 720K)	(octet)
Offset 22 :	nombre de secteurs d'une FAT	(mot)
Offset 24 :	nombre de secteurs par piste	(mot)
Offset 26 :	nombre de têtes (de faces)	(mot)

TABLE 1

Cette fonction commence par lire la FAT (celle-ci se trouve toujours à la suite du secteur de boot et sa longueur ainsi que celle du secteur boot nous ont été fournies par type_disk()) et en stocker une copie dans un tampon. C'est dans ce tampon que l'on effectuera les modifications avant de réécrire la FAT modifiée sur disque.

Adressage de la FAT

Il s'agit ensuite d'adresser correctement la FAT, sachant que les données y sont codées sur 12 bits et que chaque entrée correspond à une grappe (et non pas à un secteur). On commence par déterminer, pour chaque piste, secteur et face voulues, le numéro d'entrée FAT correspondant. C'est ce que fait la fonction get_ entree_ FAT(), qui calcule d'abord le numéro de secteur logique correspondant à ces paramètres. Les secteurs logiques sont numérotés en continu, à partir de 0 : on compte pour chaque piste d'abord les secteurs de la face 0, puis ceux de la face 1, et ainsi de suite. Pour calculer le numéro de secteur logique, on peut utiliser l'algorithme suivant :

num_sect_log = (nb_sect_par piste * ((nbre_de_faces * numéro_piste) + numéro_face)) + numéro_secteur - 1

Ainsi, pour une disquette 360K comprenant 9 secteurs par piste, le numéro de secteur logique du secteur 3 de la piste 12, face 1, sera :

(9 * ((2 * 12) + 1)) + 3 - 1 = 227

Le numéro d'entrée dans la FAT est égal au numéro de secteur logique duquel on retranche le nombre de secteurs réservés (le secteur boot, les FAT, le répertoire racine - ces différents secteurs ne pouvant recevoir de fichiers n'ont donc pas d'entrée correspondante dans la FAT). On divise le résultat obtenu par le nombre de secteurs par grappe (valeur figurant dans la structure sdisk), puisque chaque entrée correspond à une grappe. On ajoute enfin la valeur 2, puisque les 24 premiers bits (2 fois 12 bits) de la FAT sont des entrées réservées :

num_entrée_FAT = ((num_sect_log - sect_réservés) / nb_sect_par_clust)+2

Nous voilà pourvus du numéro d'entrée FAT. Il faut ensuite déterminer où écrire la valeur FF7h, sachant que notre machine ne peut manipuler directement les données sur 12 bits. C'est ce que permet l'algorithme suivant :

- obtenir un déplacement à partir du début de la FAT en multipliant le n° d'entrée FAT par 3/2 et en ne gardant que la partie entière du résultat.
- charger le mot figurant à ce déplacement.
- si le n° d'entrée est pair, écrire la valeur FF7h dans les trois quartets faibles du mot, en préservant le quartet fort:

mot = (mot & 0F000h) + 0FF7h

- si le n° d'entrée est impair, écrire la valeur FF7h dans les trois quartets forts du mot, en préservant le quartet faible :

mot = (mot & 000Fh) + 0FF70h

Une fois cette opération terminée, il nereste plus qu'à réécrire les FAT (elles sont deux) sur le disque.

L'option de vérification est quant à elle en tous points semblable à ce que décrivait A. Hoffman. La vérification du message inscrit sur le secteur 112 de la dernière piste s'effectue correctement. Le programme, tel qu'il est, a été testé sur les deux types de disquettes. Il a passé avec succès l'épreuve de DISKCOPY, ainsi que celle de deux duplicateurs bien connus (dans des versions toutefois assez anciennes).

Pour construire le programme PROTECT :

- assembler DKPROTEC.ASM
  TASM /mx dkprotec

- sous environnement intégré Turbo C :
  . compiler PROTECT.C
  . faire l'édition des liens avec un fichier-projet (PROTECT.PRJ) composé des deux lignes suivantes :
  PROTECT
  DKPROTEC.OBJ

- par ligne de commande :
  TCC -ms protect
  TLINK /c cs protect dkprotec, protect,,cs

(les différents noms de fichiers étant évidemment précédés, le cas échéant, de leur chemin d'accès).

Bibliographie :

Dave Williams « Programming Technical Reference - IBM »

EN KIOSQUE

DEBUG Magazine

HORS SERIE N° 1

"PROGRAMMES LECTEURS"

Prix : 58 F.

DEBUG magazine

CAHIER LISTINGS

SOMMAIRE

I	Programme Arabe - Romain	30
II	Notion d'Algorithmique	31
III	Génération Automatique de grilles de mots casés	34
IV	High Screen	36
V	Paradox	36
VI	Gestionnaire de périphériques MS - DOS en C	37
VII	La Gestion mémoire sous DOS	41
VIII	Un Gestionnaire de mémoire cache en C	42
IX	Protect	44

I : Programme Arabe Romain

```
{*********************************************************************}
{*                                                                   *}
{* Conversion de nombre romain en nombre arabe et réciproquement     *}
{*                                                                   *}
{* (c) Jean-Christophe RIAT                          Décembre 1990   *}
{*                                                                   *}
{* Turbo-Pascal versions 3,4 et 5                                    *}
{*                                                                   *}
{*********************************************************************}

PROGRAM Arabe_Romain;


USES Crt;  { A supprimer avec la version 3 de Turbo-Pascal }


TYPE

  Symbole_Arabe  = '0'..'9';
  Symbole_Romain = CHAR;  { Pascal ne permet pas d'être plus précis! }

  String_50 = STRING[50];

  Domaine = SET OF CHAR;  { Ensemble de caractères... }


VAR

  Termine : BOOLEAN;
  Choix   : CHAR;

{-------------------------------------------------------------------}
{ Lecture dans "Nombre" d'une chaîne de caractères entrée au clavier }
{ Vérification de l'appartenance des caractères au domaine "Liste_   }
{ Symboles" (paramètre de la fonction de type "SET OF CHAR")         }
{-------------------------------------------------------------------}
PROCEDURE Lire_Nombre(VAR Nombre : String_50;
                      Liste_Symboles : Domaine);
VAR

  Syntaxe_Ok : BOOLEAN;
  i : INTEGER;

BEGIN

  REPEAT

    { Lecture de la chaîne de caractères }
    WRITELN;
    WRITE('Entrer le nombre à convertir: ');
    READLN(Nombre);

    { Vérification de l'appartenance des caractères au domaine }
    i:=0;
    Syntaxe_Ok:=TRUE;
    WHILE Syntaxe_Ok AND (i<LENGTH(Nombre)) DO
      BEGIN
        i:=i+1;
        IF NOT(Nombre[i] IN Liste_Symboles)
          THEN
            Syntaxe_Ok:=FALSE;
      END;

  UNTIL Syntaxe_Ok AND (Nombre<>'');

END;


{-------------------------------------------------------------------}
{ Procédure de conversion d'un nombre écrit sous forme arabe en son  }
{ équivalent dans le système de numération romain                    }
{-------------------------------------------------------------------}
PROCEDURE Convertir_Arabe_en_Romain;

VAR

  Nombre_Entier : INTEGER;
  Nombre_Arabe  : String_50;
  Nombre_Ok     : BOOLEAN;
  erreur : INTEGER;  { Pour l'appel de la procédure VAL }

  { Ecriture d'un chiffre dans le système de numération romain }
  PROCEDURE Convertir_Chiffre(Chiffre : Symbole_Arabe;
                              symb1,symb2,symb3 : Symbole_Romain);
  BEGIN
    CASE Chiffre OF
      '0' : ;  { Les romains ne connaissaient pas le zéro! }
      '1' : WRITE(symb3);
      '2' : WRITE(symb3,symb3);
      '3' : WRITE(symb3,symb3,symb3);
      '4' : WRITE(symb3,symb2);
      '5' : WRITE(symb2);
      '6' : WRITE(symb2,symb3);
      '7' : WRITE(symb2,symb3,symb3);
      '8' : WRITE(symb2,symb3,symb3,symb3);
      '9' : WRITE(symb3,symb1);
    END;

  END;


BEGIN

  { Lecture du nombre à convertir }
  REPEAT

    { Lecture du nombre arabe }
    Lire_Nombre(Nombre_Arabe,['0'..'9']);

    { Vérification du nombre arabe }
    VAL(Nombre_Arabe,Nombre_Entier,erreur);
    Nombre_Ok:=(Nombre_Entier<>0) AND (Nombre_Entier<4000);
    IF Nombre_Ok
      THEN
        { Mise au format du nombre }
        WHILE LENGTH(Nombre_Arabe)<>4 DO
          Nombre_Arabe:='0'+Nombre_Arabe
      ELSE
        { Affichage d'un message d'erreur }
        WRITELN('Le nombre doit être compris entre 1 et 3999!')

  UNTIL Nombre_Ok;

  { Présentation du résultat }
  WRITE(Nombre_Entier,' = ');

  { Conversion du nombre arabe en nombre romain }
  Convertir_Chiffre(Nombre_Arabe[1],' ',' ','M');
  Convertir_Chiffre(Nombre_Arabe[2],'M','D','C');
  Convertir_Chiffre(Nombre_Arabe[3],'C','L','X');
  Convertir_Chiffre(Nombre_Arabe[4],'X','V','I');
  WRITELN;

END;



{-------------------------------------------------------------------}
{ Procédure de conversion d'un nombre représenté dans le système de  }
{ numération romain en son équivalent dans le système arabe          }
{-------------------------------------------------------------------}
PROCEDURE Convertir_Romain_en_Arabe;

VAR

  Nombre_Entier   : INTEGER;
  Nombre_Romain   : String_50;
  Derniere_Valeur : INTEGER;
  Nombre_Ok       : BOOLEAN;
  i : INTEGER;


  { Calcul de la valeur d'un symbole romain }
  PROCEDURE Convertir_Symbole(VAR Nombre_Entier : INTEGER;
                              VAR Derniere_Valeur : INTEGER;
                              Valeur_Courante : INTEGER);
  BEGIN
    IF Derniere_Valeur>Valeur_Courante
      THEN
        Nombre_Entier:=Nombre_Entier-Valeur_Courante
      ELSE
        Nombre_Entier:=Nombre_Entier+Valeur_Courante;
    Derniere_Valeur:=Valeur_Courante;
  END;


  { Vérification de l'écriture d'un digit du nombre romain }
  PROCEDURE Verifier(Nb_Romain : String_50;
                     Symb1,Symb2,Symb3 : Symbole_Romain;
                     VAR i : INTEGER);

  BEGIN

    Nb_Romain:=Nb_Romain+'$';

    { Test du premier caractere }
    IF Nb_Romain[i] IN [Symb2,Symb3] THEN
      BEGIN
```

```
        i:=i+1;

        { Test du deuxième caractère }
        IF (Nb_Romain[i]=Symb3) OR ((Nb_Romain[i-1]=Symb3)
                    AND (Nb_Romain[i] IN [Symb1,Symb2])) THEN
          BEGIN
            i:=i+1;

            { Test du troisième caractère }
            IF (Nb_Romain[i]=Symb3) AND (Nb_Romain[i-1]=Symb3) THEN
              BEGIN
                i:=i+1;

                { Test du quatrième caractère }
                IF (Nb_Romain[i]=Symb3) AND (Nb_Romain[i-3]=Symb2)
                  THEN i:=i+1;
              END;
          END;
      END;
  END;



BEGIN

  { Introduction du nombre à convertir }
  REPEAT

    { Lecture du nombre romain }
    Lire_Nombre(Nombre_Romain,['I','V','X','L','C','D','M']);

    { Verification du nombre romain }
    i:=1;
    Verifier(Nombre_Romain,' ',' ','M',i);
    Verifier(Nombre_Romain,'M','D','C',i);
    Verifier(Nombre_Romain,'C','L','X',i);
    Verifier(Nombre_Romain,'X','V','I',i);
    Nombre_Ok:=(i=LENGTH(Nombre_Romain)+1);
    IF NOT Nombre_OK
      THEN
        WRITELN('Le nombre introduit est incorrect...');

  UNTIL Nombre_Ok;

  { Conversion du nombre romain en nombre arabe }
  Nombre_Entier:=0;
  Derniere_Valeur:=0;
  FOR i:=LENGTH(Nombre_Romain) DOWNTO 1 DO
    CASE Nombre_Romain[i] OF
      'M' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,1000);
      'D' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,500);
      'C' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,100);
      'L' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,50);
      'X' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,10);
      'V' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,5);
      'I' : Convertir_Symbole(Nombre_Entier,Derniere_Valeur,1);
    END;

  { Affichage du résultat }
  WRITELN(Nombre_Romain,' = ',Nombre_Entier);

END;



{ Le corps du programme principal... }
BEGIN

  { Initialisation }
  Termine:=FALSE;

  { Traitements }
  REPEAT

    { Affichage du menu des options }
    ClrScr;
    WRITELN('TRANSCRIPTION ARABE <-> ROMAIN':55);
    WRITELN;
    WRITELN;
    WRITELN('1. Transcription Arabe -> Romain':44);
    WRITELN;
    WRITELN('2. Transcription Romain -> Arabe':44);
    WRITELN;
    WRITELN('3. Fin':17);
    WRITELN;
    WRITELN;


  { Introduction du choix de l'utilisateur }
    REPEAT
      WRITE('Votre choix: ':21);
      READLN(Choix);
    UNTIL Choix IN ['1'..'3'];

    { Traitement du choix de l'utilisateur }
    CASE Choix OF
      '1' : Convertir_Arabe_en_Romain;
      '2' : Convertir_Romain_en_Arabe;
      '3' : Termine:=TRUE;
    END;

    { Validation utilisateur pour poursuivre }
    WRITELN;
    WRITE('Appuyer une touche quelconque...':60);
    WHILE ReadKey='' DO;

  UNTIL Termine;

  { Conclusion }
  WRITELN;
  WRITELN('Au revoir...');

END.
```

II - Notion D'algorithmique

```
/*****************************************************************
 Nom fichier   : PILECHAINE.C
 Version ..... : 1.00
 Objet ....... : Implémentation d'une pile en représentation chainée
                 (par pointeur)
 Date création : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*****************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include "pilechaine.h"


/*===============================================================
====*/
/*              MYMALLOC()
*/
/*===============================================================
====*/
static char *MyMalloc( Taille )
unsigned Taille;
{
   char *Ptr;

   Ptr = (char *) malloc( Taille );
   if (Ptr == NULL)
      fprintf(stderr,"Plus de mémoire dans MyMalloc()\n");

return Ptr;
}


/*===============================================================
====*/
/*                EMPILER()                                      */
/*===============================================================
====*/
int  Empiler( pPile, Donnée )
PileChaine_t *pPile; /* Pointeur sur la structure pile */
long         Donnee; /* Donnée à placer dans la pile */
{
   Noeud_t *pPileElement;

   /*** Creation de l'élément à ajouter dans la pile ***/
   pPileElement = (Noeud_t *) MyMalloc( sizeof(Noeud_t) );
   if ( pPileElement == NULL )
      return ER_OUTOFMEM;

   /*** MAJ de l'élément est chainage avec les autres éléments ***/
   pPileElément->Donnée = Donnée;
   pPileElément->pNext  = pPile->pSommet;

   /*** MAJ de l'entête de pile ***/
   pPile->pSommet     = pPileElément;
   pPile->NbreElém++;

return ER_OK;
} /* fin de fct : Empiler() */


/*===============================================================
```

```
====*/
/*                D     E     P     I     L     E     R     (     )
*/
/*=========================================================================
====*/
int  Dépiler( pPile, pDonnee )
PileChaine_t *pPile;   /* Pointeur sur la structure pile */
long         *pDonnée; /* Donnée à sortir de la pile */
{
   int Rts = ER_OK;
   Noeud_t *pPileElément;

   if ( PileVide(pPile) )
   {
      /* La pile est vide erreur */
      *pDonnée = 0L;

      Rts = ER_PILEVIDE;
   } /* endif */
   else
   {
      /*** Récupéré le sommet de la pile ***/
      pPileElément = pPile->pSommet;
      *pDonnée     = pPileElément->Donnée;

      /*** MAJ de l'entête de la pile ***/
      pPile->NbreElem--;
      pPile->pSommet = pPileElément->pNext;

       /*** libération de la mémoire de l'élément retirer  de la pile
***/
      free( pPileElement );

      Rts = ER_OK;
   } /* endelse */

return Rts;
} /* fin de fct : Depiler() */


/*=========================================================================
====*/
/*              PILEVIDE()
*/
/*=========================================================================
====*/
BOOL PileVide( pPile )
PileChaine_t *pPile;   /* Pointeur sur la structure pile */
{
   BOOL Rts = FAUX;

   if ( pPile->NbreElem == 0 ) Rts = VRAI;

return Rts;
} /* fin de fct : PileVide() */


/************************************************************************
******
            Fin du fichier   : PILECHAINE.C
************************************************************************
*****/

#ifndef __PILECHAINE_H__
#define __PILECHAINE_H__ 1



/************************************************************************
******
 Nom fichier   : PILECHAINE.H
 Version ..... : 1.00
 Objet ....... : Définition des types de données de l'implémentation
d'une
                 pile en représentation chainée (par pointeur)
 Date création : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
************************************************************************
*****/


/*-----------------------------------------*/
/*  DEFINITION DES MACROS                  */
/*-----------------------------------------*/
#define VRAI        1
#define FAUX        0
#define ER_OK       1
#define ER_PILEVIDE -1
#define ER_OUTOFMEM -2


/*-----------------------------------------*/
/* DECLARATION DES TYPES DE DONNEES       */
/*-----------------------------------------*/
typedef int BOOL;

struct Noeud_s
{
   long           Donnée; /* Donnée à stocker dans la pile */
    struct Noeud_s *pNext; /* Pointeur sur l'élément suivant dans la
pile */
};
typedef struct Noeud_s Noeud_t;

struct PileChaine_s
{
   unsigned int NbreElem; /* Nbombre d'élément dans la pile */
   Noeud_t      *pSommet; /* Pointeur sur le sommet de la pile */
};
typedef struct PileChaine_s PileChaine_t; /* Type pile contigüe */


/*-----------------------------------------*/
/*DECLARATION DES PROTOTYPES              */
/*-----------------------------------------*/
int  Empiler( PileChaine_t *pPile, long Donnée );
int  Dépiler( PileChaine_t *pPile, long *pDonnée );
BOOL PileVide( PileChaine_t *pPile );


/************************************************************************
******
 Fin du fichier   : PILECHAINE.H
************************************************************************
*****/
#endif



/************************************************************************
******
 Nom fichier   : PILECH_T.C
 Version ..... : 1.00
 Objet ....... : Test des fonctions de PILECHAINE.C
 Date création : 18/11/90
 Date MAJ .... : 18/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
************************************************************************
*****/

#include <stdio.h>
#include "pilechaine.h"



/*=========================================================================
====*/
/*        MAIN()
*/
/*=========================================================================
====*/
/*
 * Description:
 *-----------------------------------*/
main()
{
   PileChaine_t MaPile;
   long Data;

   memset( &MaPile, 0, sizeof(PileChaine_t) );

   Empiler( &MaPile, 1 );

   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);

   Empiler( &MaPile, 1 );
   Empiler( &MaPile, 2 );
   Empiler( &MaPile, 3 );
   Empiler( &MaPile, 4 );
   Empiler( &MaPile, 5 );
   Empiler( &MaPile, 6 );

   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);
   Dépiler( &MaPile, &Data );  printf("Data %d\n",Data);

} /* fin de fct : main() */



/************************************************************************
******
 Fin du fichier   : PILECH_T.C
************************************************************************
```

```
*****/



/*************************************************************************
******
 Nom fichier   : PILECONT.C
 Version ..... : 1.00
 Objet ....... : Implementation d'une pile en representation contigue
                 (par tableau indice)
 Date création : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*************************************************************************
*****/

#include "pilecont.h"


/*=======================================================================
====*/
/*      EMPILER()
*/
/*=======================================================================
====*/
int  Empiler( pPile, Donnée )
PileCont_t *pPile; /* Pointeur sur la structure pile */
long       Donnée; /* Donnée à placer dans la pile */
{
   int Rts = ER_OK;
   int Pos = pPile->Sommet;

   Pos++;

   if ( Pos < MAXPILE )
   {
      /* Cas normal on insert une donnée dans la pile */
      pPile->Pile[Pos] = Donnée;
      pPile->Sommet    = Pos;

      Rts = ER_OK;
   } /* endif */
   else
   {
      /* On essaye d'écrire en dehors de la pile donc erreur */
      Rts = ER_OUTOFPILE;
   } /* endelse */

return Rts;
} /* fin de fct : Empiler() */


/*=======================================================================
====*/
/*      DEPILER()                                                      */
/*=======================================================================
====*/
int  Dépiler( pPile, pDonnée )
PileCont_t *pPile;
long       *pDonnée;
{
   int Rts = ER_OK;

   if ( PileVide(pPile) )
   {
      /* La pile est vide erreur */
      *pDonnée = 0;

      Rts = ER_PILEVIDE;
   } /* endif */
   else
   {
      /* Cas normal on insert une donnée dans la pile */
      *pDonnée = pPile->Pile[pPile->Sommet];
      pPile->Sommet--;

      Rts = ER_OK;
   } /* endelse */

return Rts;
} /* fin de fct : Depiler() */


/*=======================================================================
====*/
/*     PILEVIDE()                                                      */
/*=======================================================================
====*/
BOOL PileVide( pPile )
PileCont_t *pPile;
{
   BOOL Rts = FAUX;

   if ( pPile->Sommet == 0 ) Rts = VRAI;

return Rts;
} /* fin de fct : PileVide() */


/*************************************************************************
******
 Fin du fichier   : PILECONT.C
*************************************************************************
*****/



#ifndef __PILECONT_H__
#define __PILECONT_H__ 1
/*************************************************************************
********
 Nom fichier   : PILECONT.H
 Version ..... : 1.00
 Objet ....... : Définition des types de données de l'implémentation
d'une
                 pile en représentation contigüe (par tableau indice)
 Date création : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*************************************************************************
*****/

/*-----------------------------------------------------*/
/* DEFINITION DES DEFINES                              */
/*-----------------------------------------------------*/
#define MAXPILE 5           /* Taille maximum de la pile */

/*-----------------------------------------------------*/
/* DEFINITION DES MACROS                               */
/*-----------------------------------------------------*/
#define VRAI         1
#define FAUX         0
#define ER_OK        1
#define ER_PILEVIDE  -1
#define ER_OUTOFPILE -2

/*-----------------------------------------------------*/
/* DECLARATION DES TYPES DE DONNEES                    */
/*-----------------------------------------------------*/
typedef int BOOL;
struct PileCont_s
{
   unsigned int Sommet;              /* Sommet de la pile */
   long         Pile[MAXPILE];       /* La pile elle même */
};

typedef struct PileCont_s PileCont_t; /* Type pile contigue */

/*-----------------------------------------------------*/
/* DECLARATION DES PROTOTYPES                          */
/*-----------------------------------------------------*/
int  Empiler( PileCont_t *pPile, long Donnée );
int  Depiler( PileCont_t *pPile, long *pDonnée );
BOOL PileVide( PileCont_t *pPile );


/*************************************************************************
******
 Fin du fichier   : PILECONT.H
*************************************************************************
*****/
#endif



/*************************************************************************
******
 Nom fichier   : PILEC_T.C
 Version ..... : 1.00
 Objet ....... : Test des fonctions de PILECONT.C
 Date création : 15/11/90
 Date MAJ .... : 15/11/90
 Auteur ...... : Frederic Berton
 Environnement : Microsoft C / Zortech C / Turbo C / UNIX V
*************************************************************************
*****/

#include <stdio.h>
#include <conio.h>
#include "pilecont.h"

void TRACEPILE( pPile )
PileCont_t *pPile;
{
   char Rep;
   int i;

   printf("Trace pile (o/n) : "); Rep = getche(); putchar('\n');
   if ( Rep == 'o' )
   {
      printf("Sommet   = %d\n",pPile->Sommet);
      for (i=1; i<MAXPILE; i++)
```

```
      {
         printf("Pile[%d] = %d\n",i,pPile->Pile[i]);
      } /* endfor */
   } /* endif */
}

/*=================================================================
====*/
/*  MAIN()                                                        */
/*=================================================================
====*/
/*
 * Description:
 *-------------------------------------*/
main()
{
   PileCont_t MaPile;
   long Data;

   memset( &MaPile, 0, sizeof(PileCont_t) );

   Empiler( &MaPile, 1 );     TRACEPILE( &MaPile );
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
   Empiler( &MaPile, 2 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 3 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 4 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 5 );     TRACEPILE( &MaPile );
   Empiler( &MaPile, 6 );     TRACEPILE( &MaPile );
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);
    Dépiler( &MaPile, &Data ); TRACEPILE( &MaPile ); printf("Data
%d\n",Data);

} /* fin de fct : main() */



/*************************************************************************
******
 Fin du fichier   : PILEC_T.C
*************************************************************************
*****/
```

III - Génération automatique de grilles de mots casés

```
{-------------------------------------------------------}
{  GENERATION AUTOMATIQUE DE GRILLES DE MOTS CASES      }
{  AVEC 3,5,7 ou 9 LETTRES                              }
{                                                       }
{  (c) 6/90  -  CHAMBARET Laurent                       }
{  pour DEBUG MAGAZINE                                  }
{-------------------------------------------------------}

{$M $2000,0,655360}
{$A+,E-,N-,D+,L+,O-}
{$B-,F-,I+,R-,S+,V-}
Program  MotsCases;
Uses  Crt;

Const
  X0 = 30;
  Y0 = 5;
  ExtFicGrilles = '.GRI';
  ExtMots       = '.MOT';
  MaxMots       = 6000;
  MaxLgMot      = 9;
Type
  TCptMot   = 0..MaxMots;
  TCptCar   = 0..MaxLgMot;
  TStrMot   = String[MaxLgMot];
  TMots     = Array [1..MaxMots] Of TStrMot;
  TLettres  = Array ['A'..'Z'] Of
                Record
                  M  : TCptMot;
                  Nb : TCptMot;
                End;
  TGrilleXY = Array [1..MaxLgMot] Of TStrMot;
Var
  Mot           : TMots;
  NbMots,
  PremierMot    : TCptMot;
  LgMot         : TCptCar;
  LgMotCar      : Char;
  Lettre        : TLettres;
  GrilleX,
  GrilleY       : TGrilleXY;
  NbGrilles,
  XGrilleOk     : Word;
  AfficherRech,
  Enregistrer   : Boolean;

FicGrilles    : Text;


Procedure  Beep;
  Begin
    Sound(100);
    Delay(30);
    NoSound;
  End;

Procedure  TestESC;
  Begin
    If KeyPressed And (ReadKey=#27) Then
      Begin
        If Enregistrer Then Close(FicGrilles);
        GotoXY(1,24);
        Halt(0);
      End;
  End;

Procedure  LireLeFichierDesMots;
  Var
    Fic : Text;
    L   : Char;
  Begin
    NbMots := 0;
    For L:='A' To 'Z' Do
      Begin
        Lettre[L].M := 0;
        Lettre[L].Nb := 0;
      End;
    Assign(Fic,LgMotCar+ExtMots);
    {$I-}
    Reset(Fic);
    {$I+}
    If IoResult=0 Then
      Begin
        While (Not Eof(Fic)) And (NbMots<>MaxMots) Do
          Begin
            Inc(NbMots);
            ReadLn(Fic,Mot[NbMots]);
            L := UpCase(Mot[NbMots][1]);
            If Lettre[L].M=0 Then Lettre[L].M := NbMots;
            Inc(Lettre[L].Nb);
          End;
        Close(Fic);
      End;
  End;

Procédure  InitialiserLaGrille;
  Var
    XY : TCptCar;
  Begin
    For XY:=1 To LgMot Do
      Begin
        GrilleX[XY] := '';
        GrilleY[XY] := '';
      End;
  End;

Procédure  AfficherLeCadreDeLaGrille;
  Var
    X, Y : TCptCar;
  Begin
    TextColor(Yellow);
    GotoXY(X0,Y0);
    Write(#218+#196);
    For X:=1 To LgMot*2 Do Write(#196);
    Write(#191);
    For Y:=1 To LgMot Do
      Begin
        GotoXY(X0,Y0+Y);
        Write(#179);
        GotoXY(X0+2+LgMot*2,Y0+Y);
        Write(#179);
      End;
    GotoXY(X0,Y0+LgMot+1);
    Write(#192+#196);
```

```
    For X:=1 To LgMot*2 Do Write(#196);
    Write(#217);
    TextColor(Blue);
    For Y:=1 To LgMot Do
      If Not Odd(Y) Then
        For X:=1 To LgMot Do
          If Not Odd(X) Then
            Begin
              GotoXY(X0+X*2,Y0+Y);
              Write(#219);
            End;
  End;

Procédure  EffacerLaGrille;
  Var
    X, Y : TCptCar;
  Begin
    For Y:=1 To LgMot Do
      For X:=1 To LgMot Do
        If Odd(Y) Or Odd(X) Then
          Begin
            GotoXY(X0+X*2,Y0+Y);
            Write(' ');
          End;
  End;


Procédure  AfficherLePremierMot( M : TCptMot );
  Var
    X : TCptCar;
  Begin
    For X:=1 To LgMot Do
      Begin
        GotoXY(X0+X*2,Y0+1);
        Write(Mot[M][X]);
      End;
  End;

Procédure  AfficherLaGrille;
  Var
    X, Y : TCptCar;
  Begin
    EffacerLaGrille;
    TextColor(Yellow);
    For Y:=1 To LgMot Do
      If Odd(Y) And (GrilleY[Y]<>'') Then
        For X:=1 To LgMot Do
          Begin
            GotoXY(X0+X*2,Y0+Y);
            Write(GrilleY[Y][X]);
          End;
    For X:=1 To LgMot Do
      If Odd(X) And (GrilleX[X]<>'') Then
        For Y:=1 To LgMot Do
          Begin
            GotoXY(X0+X*2,Y0+Y);
            Write(GrilleX[X][Y]);
          End;
  End;

Procédure  AfficherLeNombreDeMotsEtudies( M : TCptMot );
  Begin
    TextColor(LightCyan);
    GotoXY(17,7);
    Write(M:4);
  End;

Procédure  AfficherLeNombreDeGrillesCompletes;
  Begin
    TextColor(LightCyan);
    GotoXY(21,8);
    Write(NbGrilles);
  End;

Procédure  EnregistrerLaGrille;
  Var
    X, Y : TCptCar;
  Begin
    For Y:=1 To LgMot Do
      If Odd(Y) Then
        WriteLn(FicGrilles,GrilleY[Y])
       Else
        For X:=1 To LgMot Do
          Begin
            If Odd(X) Then
              Write(FicGrilles,GrilleX[X][Y]+' ');
            If X=LgMot Then WriteLn(FicGrilles);
          End;
    WriteLn(FicGrilles);
  End;

Procédure  GrilleComplète;
  Var
    X, Y : TCptCar;
  Begin
    Beep;
    Inc(NbGrilles);
    AfficherLeNombreDeGrillesComplètes;
    TextColor(Yellow);
    Inc(XGrilleOk);
    If XGrilleOk*(LgMot+2)>80 Then XGrilleOk := 1;
    For Y:=1 To LgMot Do
      If Odd(Y) Then
        Begin
          GotoXY(1+(XGrilleOk-1)*(LgMot+2),24-LgMot+Y);
          Write(GrilleY[Y]);
        End;
    For X:=1 To LgMot Do
      If Odd(X) Then
        For Y:=1 To LgMot Do
          Begin
            GotoXY((XGrilleOk-1)*(LgMot+2)+X,24-LgMot+Y);
            Write(GrilleX[X][Y]);
          End;
    If Enregistrer Then EnregistrerLaGrille;
  End;

Procédure  TesterMotCorrect( Var Grille : TGrilleXY;
                                 XYn    : TCptCar;
                                 M      : TCptMot;
                             Var Bon    : Boolean );
  Var
    XY : TCptCar;
  Begin
    Bon := True;
    XY := 0;
    While Bon And (XY<>LgMot) Do
      Begin
        Inc(XY);
        If Odd(XY) And (Grille[XY]<>'') And
          (Grille[XY][XYn]<>Mot[M][XY]) Then Bon := False;

      End;
  End;

Procédure  TesterMotEnDouble(     XY  : TCptCar;
                                  M   : TCptMot;
                             Var Bon : Boolean );
  Var
    X, Y : TCptCar;
  Begin
    For X:=1 To XY Do
      If Odd(X) And (GrilleX[X]=Mot[M]) Then Bon := False;
    For Y:=1 To XY Do
      If Odd(Y) And (GrilleY[Y]=Mot[M]) Then Bon := False;
  End;

Procédure  ExplorerY( Y : TCptCar );  Forward;

Procédure  ExplorerX( X : TCptCar );
  Var
    L   : Char;
    M   : TCptMot;
    Bon : Boolean;
  Begin
    L := UpCase(GrilleY[1][X]);
    For M:=Lettre[L].M To Lettre[L].M+Lettre[L].Nb-1 Do
      Begin
        TesterMotCorrect(GrilleY,X,M,Bon);
        TesterMotEnDouble(X,M,Bon);
        If Bon Then
          Begin
            GrilleX[X] := Mot[M];
            If AfficherRech Then AfficherLaGrille;
            If X=LgMot Then GrilleComplete
                       Else ExplorerY(X+2);
            GrilleX[X] := '';
          End;
      End;
  End;

Procédure  ExplorerY( Y : TCptCar );
  Var
    L      : Char;
    M,
    M1, M2 : TCptMot;
    Bon    : Boolean;
  Begin
    If GrilleX[1]='' Then
      Begin
        M1 := PremierMot;
        M2 := NbMots
      End
     Else
      Begin
        L := UpCase(GrilleX[1][Y]);
        M1 := Lettre[L].M;
        M2 := M1+Lettre[L].Nb-1;
      End;
    For M:=M1 To M2 Do
      Begin
        If Y=1 Then
          Begin
            AfficherLeNombreDeMotsEtudies(M);
```

```
            If Not AfficherRech Then
              AfficherLePremierMot(M);
          End;
        TesterMotCorrect(GrilleX,Y,M,Bon);
        TesterMotEnDouble(Y,M,Bon);
        If Bon Then
          Begin
            GrilleY[Y] := Mot[M];
            If AfficherRech Then AfficherLaGrille;
            ExplorerX(Y);
            GrilleY[Y] := '';
          End;
        TestESC;
      End;
  End;

Procédure  ChercherDesGrilles;
  Begin
    XGrilleOk := 0;
    NbGrilles := 0;
    AfficherLeNombreDeGrillesCompletes;
    AfficherLeCadreDeLaGrille;
    InitialiserLaGrille;
    ExplorerY(1);
    EffacerLaGrille;
  End;


Var
  C   : Char;
  St  : String;
  Err : Integer;
Begin
  ClrScr;
  TextColor(LightRed);
  WriteLn('GENERATION DE GRILLES DE MOTS CASES');

  TextColor(Yellow);
  Write('Nombre de lettres (3,5,7,9) ? ');
  Repeat
    C := ReadKey;
  Until C In ['3','5','7','9'];

  WriteLn(C);
  LgMotCar := C;
  LgMot := Ord(C)-Ord('0');

  Write('Afficher les recherches (O,N) ? ');
  Repeat
    C := UpCase(ReadKey);
  Until C In ['O','N'];
  WriteLn(C);
  AfficherRech := (C='O');

  Write('Enregistrer les solutions (O,N) ? ');
  Repeat
    C := UpCase(ReadKey);
  Until C In ['O','N'];
  WriteLn(C);
  Enregistrer := (C='O');
  If Enregistrer Then
    Begin
      Assign(FicGrilles,LgMotCar+ExtFicGrilles);
      Rewrite(FicGrilles);
    End;

  LireLeFichierDesMots;
  WriteLn('Nombre de mots :',NbMots:4);

  Write('Premier mot ? ');
  Repeat
    ReadLn(St);
    If St='' Then St := '1';
    Val(St,PremierMot,Err);
    If Not(PremierMot In [1..NbMots]) Then Err := 1;
    If Err<>0 Then
      Begin
        Beep;
        GotoXY(15,6);
        Write('      ');
        GotoXY(15,6);
      End;
  Until Err=0;

  WriteLn('Mot étudié     :');
  WriteLn('Grilles complètes :');
  WriteLn('Arrêt avec ESC.');
  ChercherDesGrilles;
  If Enregistrer Then Close(FicGrilles);
  GotoXY(1,24);
End.
```

IV - High Screen 5

```
{Source généré par HSGen le 05-04-1991 à 21:27:27}
{Modèle   : DEMO}
{Type     : Saisie d'un écran pleine page.}
{Langage  : Turbo Pascal}
{Interface: Linké}

Program DEMO;
Uses HSTPL;

  {...Déclaration des variables.}
Var
  REF1:Integer;
  CLIENT1:String;
  QTE1:Integer;
  PRIX1:Real;
  TOTAL1:Real;

Begin
  { ...Squelette de gestion de la saisie pleine page}
  AppelHS('Option,Police,Normal');
  AppelHS('Scrolling,Monte,1,1,60,80,0');
  AppelHS('Utilise,DEMO.AID');

  AppelHS('Affiche,REF1='+ITOS(REF1,3));
  AppelHS('Affiche,CLIENT1='+CLIENT1);
  AppelHS('Affiche,QTE1='+ITOS(QTE1,2));
  AppelHS('Affiche,PRIX1='+RTOS(PRIX1,6,2));
  AppelHS('Affiche,TOTAL1='+RTOS(TOTAL1,9,2));

  { ...Boucle principale.}
  AppelHS('Souris,Oui');
  AppelHS('Sortie,ESC=A,RET=S,TF0=V');
  HsTouche:='?';

  While(HsTouche<>'ESC') And(HsTouche<>'TF0')Do
    Begin
      AppelHS('Ecran,Saisie/CR');
    End;
  If HsTouche<>'ESC' Then
    Begin
      {  ...Récupération des valeurs saisies.}
      AppelHS('Recupere,REF1');
      REF1:=HsEntier;

      AppelHS('Recupere,CLIENT1');
      CLIENT1:=HsChaine;
      AppelHS('Recupere,QTE1');
      QTE1:=HsEntier;
      AppelHS('Recupere,PRIX1');
      PRIX1:=HsReel;
      AppelHS('Recupere,TOTAL1');
      TOTAL1:=HsReel;
    End;
  AppelHS('Scrolling,Monte,1,1,60,80,0');
End.
```

V - Paradox

```
/* Paradox Engine Library test */

/* tous les identificateurs commmençant par PX... ou px... sont
   prédéfinis par Paradox Engine dans pxengine.h.
   Ce programme lit une table dénommée "BACH.DB" et affiche à l'écran
   sa structure et le contenu de tous les enregistrements */

#include <stdlib.h>
#include <string.h>
#include <pxengine.h>

#undef DEBUG

#define TABLENAME  "bach"                    /* nom de la table */
#define MAXFLDNAMESIZE      25
#define MAXFLDTYPESIZE      4

typedef struct
{
          char      fldName[MAXFLDNAMESIZE+1];
          char      fldType[MAXFLDTYPESIZE+1];
} DBSTRUCTURE;                  /* cette structure contiendra les */
```

```
                    /* noms et les types des champs */

TABLEHANDLE                 tblHandle;
RECORDNUMBER        nRecs;
int                                 nFlds;
DBSTRUCTURE                 *dbStructure;

int openDataBase()
{
          int       pxErr;

          if((pxErr=PXTblOpen(TABLENAME,&tblHandle,0,FALSE))!=PXSUCCESS)
                    return(pxErr);
          /* Lecture du nombre d'enregistrements */
          if((pxErr=PXTblNRecs(tblHandle,&nRecs))!=PXSUCCESS)
                    return(pxErr);
          /* Lecture du nombre de champs */
          if((pxErr=PXRecNFlds(tblHandle,&nFlds))!=PXSUCCESS)
                    return(pxErr);
          printf("Table has %ld records with %d fields
each\n",nRecs,nFlds);
          return(PXSUCCESS);
}

int listDBFieldType()
{
          int                               pxErr,i;
          char
fldName[MAXFLDNAMESIZE+1],fldType[MAXFLDTYPESIZE+1];
          DBSTRUCTURE         *ptr;

          if(nFlds>0)
          {
                    /* allocation d'un tableau de taille suffisante pour
contenir les
                       noms et types de tous les champs */

if((dbStructure=(DBSTRUCTURE*)malloc(sizeof(DBSTRUCTURE)*nFlds))==NULL)
                                  return(PXERR_OUTOFMEM);
                    printf("\nThe fields are:\n");
                    for(i=1;i<=nFlds;i++)
                    {
                                  /* Lecture du nom du i-ème champ */

if((pxErr=PXFldName(tblHandle,(FIELDHANDLE)i,MAXFLDNAMESIZE,fldName))!=
PXSUCCESS)
                                            return(pxErr);
                                  /* Lecture du type du i-ème champ */

if((pxErr=PXFldType(tblHandle,(FIELDHANDLE)i,MAXFLDTYPESIZE,fldType))!=
PXSUCCESS)
                                            return(pxErr);
                                  printf("Field %u: %-25s %-
4s\n",i,fldName,fldType);
                                  ptr=&(dbStructure[i-1]);
                                  /* le nom et le type sont sauvegardés dans
le tableau */
                                  strcpy(ptr->fldName,fldName);
                                  strcpy(ptr->fldType,fldType);
                    }
#ifdef DEBUG
                    for(i=1;i<=nFlds;i++)
                    {
                                  ptr=&(dbStructure[i-1]);
                                  printf("Field %u: %-25s %-4s\n",i,ptr-
>fldName,ptr->fldType);
                    }
#endif
          }
          return(PXSUCCESS);
}

int listDBRecords()
{
          int                                         pxErr,i,j;
          RECORDHANDLE        recHandle;
          DBSTRUCTURE                 *ptr;
          char                                buffer[256];

          if(nRecs>0)
          {
                    printf("\nListing of database records:\n");
                    /* Création d'un tampon de lecture */

if((pxErr=PXRecBufOpen(tblHandle,&recHandle))!=PXSUCCESS)
                                  return(pxErr);
                    /* déplacement du pointeur sur le premier
enregistrement */
                    if((pxErr=PXRecFirst(tblHandle))!=PXSUCCESS)
                                  return(pxErr);
                    for(i=1;i<=nRecs;i++)
                    {
                                  printf("\nRecord %3u\n",i);
                                  /* Lecture de l'enregistrement courant */

if((pxErr=PXRecGet(tblHandle,recHandle))!=PXSUCCESS)
                                            return(pxErr);
                                  for(j=1;j<=nFlds;j++)
                                  {
                                            ptr=&(dbStructure[j-1]);
                                            switch(ptr->fldType[0])
                                            {
/* La table en question ne contient que des champs alphanumériques;
ce sont donc les seuls à être traités */
                                                      case 'A':
if((pxErr=PXGetAlpha(recHandle,(FIELDHANDLE)j,255,buffer))!=PXSUCCESS)

return(pxErr);

printf("%-25s: %s\n",ptr->fldName,buffer);

break;
                                                      case 'D': /* date */
                                                      case 'S': /* entier Short */
                                                      case '$': /* Monnaie */
                                                      case 'N': /* virgule flottante */

break;
                                                      }
                                            }
                                            if(i<nRecs)
                                            /* passage à l'enregistrement suivant */

if((pxErr=PXRecNext(tblHandle))!=PXSUCCESS)
                                                      return(pxErr);
                                  }
                    }
                    /* fermeture du tampon de lecture */
                    return(PXRecBufClose(recHandle));
}

int main()
{
          int       pxErr;

          dbStructure=(DBSTRUCTURE*)NULL;

          /* La session commence par l'initialisation de Paradox Engine
*/
          if((pxErr=PXInit())!=PXSUCCESS)
          {
                    printf("%s\n",PXErrMsg(pxErr));
                    exit(pxErr);
          }
          /* Ouverture du fichier contenant la table et ses index */
          if((pxErr=openDataBase())!=PXSUCCESS)
                    printf("%s\n",PXErrMsg(pxErr));
          else
          {
                    /* Affichage des données */
                    if((pxErr=listDBFieldType(tblHandle))!=PXSUCCESS)
                              printf("%s\n",PXErrMsg(pxErr));
                    if((pxErr=listDBRecords(tblHandle))!=PXSUCCESS)
                              printf("%s\n",PXErrMsg(pxErr));
                    /* Fermeture du fichier */
                    if((pxErr=PXTblClose(tblHandle))!=PXSUCCESS)
                              printf("%s\n",PXErrMsg(pxErr));
}
          if(dbStructure!=NULL)
                    free(dbStructure);
          return(PXExit());
}
```

VI - Gestionnaire de périphériques MS - DOS en C

```
Gestionnaires de périphériques MS-DOS en C

[LISTING UN]

;  prndrv.asm
;        code d'en-tête du gestionnaire de périphérique
;copyright (c) Andy Klein 1987


PRIVATE_STACK_SIZ  equ  256   ;taille de pile privée, probablement
                              ;trop grande


IS_CHAR_DEV  equ 32768        ;bit à positionner pour ce gestionnaire
```

```
codeseg    segment para public 'CODE'
codeseg    ends

dataseg    segment para public 'DATA'
dataseg    ends


    assume    cs:codeseg, ds:dataseg, es:dataseg, ss:dataseg


codeseg    segment para public 'CODE'

    org    0        ;le code commence toujours à 0 pour un gestionnaire
                    ;avec l'en-tête en premier

public prn_driver, dev_strategy_, dev_interrupt_


prn_driver    proc    far  ;le DOS appelle le gestionnaire avec des
                           ;appels FAR

        ;début de l'en-tête

next_dev   db 4 dup(255)        ;un seul gestionnaire dans ce fichier
                                ;(long)-1 = 4 octets à 255
attribute  dw    IS_CHAR_DEV
strategy   dw    dev_strategy_  ;point d'entrée de la routine de
stratégie
interrupt  dw    dev_interrupt_ ;point d'entrée de la routine
d'interruption
dev_name   db    'PRN     '

        ;fin de l'en-tête


    ;variables dans le segment de code, adressables avant d'installer
    ;les segments de pile et de donnée
req_hdr_seg  dw    (0)    ; pointeur sur l'en-tête de requête, segment
req_hdr_off  dw    (0)    ; pointeur sur l'en-tête de requête, offset
caller_ss    dw    (0)    ; ss appelant
caller_sp    dw    (0)    ; sp appelant


        ;stratégie - cette routine de stratégie sauve un pointeur
        ;long sur un en-tête de requête dans les variables
        ; req_hdr_seg et req_hdr_off du code segment
        ; Cette adresse est passée dans lmes regsitres es:bx

dev_strategy_:

    mov word ptr cs:req_hdr_off,bx     ;sauve le pointeur
    mov word ptr cs:req_hdr_seg,es     ;dans les variables du code
   ment
    ret                                ;et retour à DOS


  blic $cswt, $begin    ; pour faire plaisir au linker de l'Aztec C.

    $cswt proc near
    $cswt endp

    $begin proc near
    $begin endp


    extrn    driver_functions_:near

    ;interruption - ce n'est pas une vraie routine d'interruption
    ; car elle se termine par un ret et non un iret.
    ; Cette fonction ne reçoit pas de paramètres, mais utilise
    ; les informations stockées dans l'en-tête de requête
    ; dont la routine de stratégie à reçu et sauvegardé
    ; un pointeur.


dev_interrupt_:
            ;ETAPE 0
            ; Sauvegarde l'état du processeur
    cli             ; Pas d'interruptions
    push ds         ; sauve les registres dans la pile
    push es
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp
    pushf
    mov cs:caller_ss,ss     ; sauve la pile de l'appelant (ss et sp)
    mov cs:caller_sp,sp
    sti                     ;autorise interruptions

            ;Le vrai boulot commence ici

            ;ETAPE 1
            ; Lit le segment du gestionnaire et le range dans ax.
            ; C'est le début de l'établissement de l'adressabilité

    mov ax,cs

            ;ETAPE 2
            ;  Recopie l'en-tête de requete pointé par
req_hdr_seg:req_hdr_off
            ;  dans notre espace adressable privé et accessible aux
            ;  fonctions C

    mov si,cs:req_hdr_off        ;charge l'adresse source
segment:offset
    mov bx,cs:req_hdr_seg        ;dans les registres ds:si
    mov ds,bx
    mov es,ax                    ;charge l'adresse destination
segment:offset
    mov di,offset cs:driver_rh_  ;dans les registres es:di
    cld                          ;flag de direction = incrémenter si et
di
    xor cx,cx                    ;efface cx
    mov cl,[si]                  ;1er octet de l'en-tête = sa longueur
    rep movsb                    ; ... et recopier ...


            ;ETAPE 3
            ;  Cette étape établit l'adressibilité du data segment
            ;  et initialise la pile du gestionnaire
            ;  sp est recopié dans bp, indispensable pour appeler
            ;  des fonctions C
            ;       ax contient toujours le segment du gestionnaire
            ;       ds, es, ss reçoivent la valeur de cs ... modèle
8080 ??

    mov bx,offset c_stack_top    ;cela va dans sp
    cli                 ;pas d'interruptions pendant les transferts de
registeres
    mov ds,ax
    mov es,ax
    mov ss,ax           ;nouvelle pile
    mov sp,bx
    mov bp,bx           ;bp = sp
    sti                 ;autorise interruptions
           ;'environnement C est prêt


           ;ETAPE 4
           ; Appel de la 1ere fonction C
           ; en passant le code de commande de l'en-tête
           ; de requête comme paramètre.
           ; (void)driver_functions(cmd_code);

    xor ax,ax                    ;efface ax
    mov bx,offset driver_rh_     ;bx pointe sur l'en-tête de requete
    mov al,byte ptr [bx + 2]     ;3rd octet de l'entete = code de
commande
    push ax                      ;empiler le paramètre pôur la fonction
C
    call driver_functions_       ;appel de la fonction
    pop ax                       ;supprime le paramètre empilé


           ;ETAPE 5
           ; A ce moment, la tâche du gestionnaire est terminée.
           ; Il faut recopier l'en-tête privé dans l'en-tête DOS
original

    mov es,cs:req_hdr_seg        ;adresse entete DOS dans es:di
    mov di,cs:req_hdr_off
    mov si,offset driver_rh_     ;ds:si  la copie de l'entete modifée
    cld                          ;direction = incrémentement
    xor cx,cx                    ;efface cx
    mov cl,[si]                  ;longueur = 1er octet
    rep movsb                    ; ... copie ...


           ;ETAPE 6
           ; Restaure l'état du processeur sauve à l'étape 0
           ; et retour au DOS

    cli
    mov ax,cs:caller_ss
    mov ss,ax
    mov ax,cs:caller_sp
    mov sp,ax
    popf
    pop bp
    pop di
    pop si
    pop dx
    pop cx
    pop bx
    pop ax
    pop es
    pop ds
    sti
    ret
```

```
prn_driver    endp          ;fin du code du gestionnaire


codeseg    ends



    ;Data segment, contient la pile locale,
    ;et la copie de l'en-tête de requête

dataseg  segment para public 'DATA'

    public c_stack_top

    db  PRIVATE_STACK_SIZ dup (0)
c_stack_top  label  word

    public  driver_rh_
driver_rh_  db  32 dup (0)   ;en tête de requête adressable

dataseg    ends


END
```

[LISTING DEUX]

```
/**  rh.h
    Définition de la structure de l'en-tête de requête
    et des define du mot d'état
copyright (c) 1987 Andy Klein
**/


typedef struct
{
    unsigned char length,        /* longueur de l'entete */
                  unit,          /* code unité */
                  cmd;           /* commande à éxécuter */
    unsigned int  status;        /* status */
    unsigned char reserved[8],
                  media_type;    /* descripteur media, périphérique bloc
seult */
    unsigned int  xfer_buf_offset,
                  xfer_buf_segment,
                  xfer_count;
    char          dummy[32 - 20];
} request_hdr;


#define  ERROR_MASK          32768
#define  BUSY_MASK           1024
#define  DONE_MASK           512

#define  WRITE_PROTECTED     0x00
#define  UNKNOWN_UNIT        0x01
#define  DEV_NOT_READY       0x02
#define  UNKNOWN_CMD         0x03
#define  CRC_ERROR           0x04
#define  BAD_DRIVE_REQ_LEN   0x05
#define  SEEK_ERROR          0x06
#define  UNKOWN_MEDIA        0x07
#define  SECTOR_NOT_FOUND    0x08
#define  PRN_NO_PAPER        0x09
#define  WRITE_FAULT         0x0A
#define  READ_FAULT          0x0B
#define  GENERAL_FAILURE     0x0C
#define  INVALID_DISK_CHG    0x0F
```

[LISTING TROIS]

```
/**  drvfunc.c
       Getsionnaire de périphérique pour DOS 2.x 3.x
       Ceci est la fonction est appelée par la fonction
       d'interruption une fois l'adressabilité établie
copyright (c) 1987 Andy Klein
**/
#define LAST_FUNCTION 15
#define Q_FUNCTIONS   LAST_FUNCTION + 1

extern void init(), output_status(), output(), output_flush(),
    output_verf(), bad_cmd();

    /**
        function_table est un tableau de pointeurs de fonctions
        retournant un type void.
    **/
void (* function_table[Q_FUNCTIONS])() = {
    /* 0 */        init,
    /* 1  */    bad_cmd,            /* test media */
    /* 2  */    bad_cmd,            /* bpb */
    /* 3  */    bad_cmd,            /* entrée ioctl */
    /* 4  */    bad_cmd,            /* entrée */
    /* 5  */    bad_cmd,            /* entrée sans attente */
    /* 6  */    bad_cmd,            /* status d'entrée */
    /* 7  */    bad_cmd,            /* balaye l'entrée */
    /* 8  */    output,
    /* 9  */    output_verf,
    /* 10 */    output_status,
    /* 11 */    output_flush,
    /* 12 */    bad_cmd,            /* sortie ioctl */
    /* 13 */    bad_cmd,            /* ouvre périphérique */
    /* 14 */    bad_cmd,            /* ferme périphérique */
    /* 15 */    bad_cmd             /* media extractible */
};


void
driver_functions(cmd)
int cmd;
{
    if ( cmd > LAST_FUNCTION )
        (void) bad_cmd();
    else
        (void) (* function_table[cmd])();
}/* driver_functions() */
```

[LISTING QUATRE]

```
/**  error.c
     Gestionnaire d'erreurs
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

void
bad_cmd()
{
    driver_rh.status = ERROR_MASK | UNKNOWN_CMD;
}


#define TIME_OUT  1
#define IO_ERR    8
#define NO_PAPER  32
#define BUSY      128


void
driver_error(stat)
int stat;
{
    int err_code;

    if ( stat & ERROR_MASK )
        stat ^= ERROR_MASK;
    switch ( stat )
    {
        case TIME_OUT:    err_code = DEV_NOT_READY;
                          break;
        case IO_ERR:      err_code = GENERAL_FAILURE;
                          break;
        case NO_PAPER:    err_code = PRN_NO_PAPER;
                          break;
        case BUSY:        err_code = DEV_NOT_READY;
                          break;
        default:          err_code = GENERAL_FAILURE;
                          break;
    }
    driver_rh.status = ERROR_MASK | err_code;
}/* driver_error() */
```

[LISTING CINQ]

```
/**  init.c
       Initialisation du gestionnaire d'imprimante
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

char *title = "\nGestionnaire d'imprimante Okidata 92";
char *copyw  = "copyright 1987 (c) Andy Klein\n";


void
init()
{
```

```
    extern unsigned _Uend;
              /* _Uend est inséré par le linker de l'Aztec C */
    unsigned show_cs();
    void reset_printer(), initialize_oki92();

    puts(title);
    puts(copyw);
    (void) reset_printer();
    (void) initialize_oki92();
    driver_rh.xfer_buf_segment = show_cs();
    driver_rh.xfer_buf_offset = (unsigned int)&_Uend;
    driver_rh.status = DONE_MASK;
} /* init() */


#define ELITE_FONT  28

void
initialize_oki92()
{
    char_2_prn(ELITE_FONT);
}/* initialize_oki92() */
```

[LISTING SIX]

```
;  show_cs.asm
;    l'appel en C est : (unsigned) show_cs();
;    Retourne la valeur du registre cs
;copyright (c) 1987 Andy Klein

include lmacros.h

procdef        show_cs
    mov ax, cs
    pret
    pend show_cs

finish
```

[LISTING SEPT]

```
/**  output.c
        Fonctions de sortie du gestionnaire d'imprimante
copyright (c) 1987 Andy Klein
**/
#include "rh.h"

extern request_hdr driver_rh;

void
output()
{
    int stat;
    unsigned  xfer_off, bytes_xferd;
    register char outch;
    char fetch_char();

    xfer_off = driver_rh.xfer_buf_offset;
    for ( bytes_xferd = 0; bytes_xferd < driver_rh.xfer_count;
            ++bytes_xferd, ++xfer_off )
    {
        outch = fetch_char(driver_rh.xfer_buf_segment, xfer_off);
        if ( (stat = char_2_prn(outch)) )
        {
            (void) driver_error(stat);
            break;
        }
    }

    driver_rh.xfer_count = bytes_xferd;
    if ( ! stat )
        driver_rh.status = DONE_MASK;
    else
        driver_rh.status = stat;
}/* output() */


void
output_verf()
{
    (void) output();
}/* out_verf() */



void
output_status()
{
    /**
        si le périphérique effectue une opération
        driver_rh.status = BUSY_MASK & DONE_MASK;
        sinon, si une écriture peut commencer imémdiatement;,
        driver_rh.status = 0 & DONE_MASK;
    **/
    if ( printer_busy() )
        driver_rh.status = BUSY_MASK & DONE_MASK;
    else
        driver_rh.status = DONE_MASK;
}/* output_status() */



void
output_flush()
{
    /**
        termine toutes les opérations en cours
        sans objet pour ce périphérique
    **/
    driver_rh.status = DONE_MASK;
}/* output_flush() */
```

[LISTING HUIT]

```
;  prlport.asm
;    Fonctions d'écriture sur le port parallèle de bas niveau
;copyright (c) Andy Klein 1987


include lmacros.h       ; macros d'appel Aztec C pour l'assembleur
                        ; pour d'autres environnements, remplacer
                        ; avec les conventions de votre compilateur


ERROR_MASK   equ    32768
PRINTER_INT  equ    017H
PRINTER_ID   equ    0            ;lpt1 = 0, lpt2 = 1, lpt3 = 2

    ;commandes imprimante, dans ah
PRINT_CHAR   equ    0
INIT_PRN     equ    1
READ_STAT    equ    2

    ;code erreur d'imprimante, retourné dans ah
TIME_OUT     equ    1
IO_ERR       equ    8
NO_PAPER     equ    32
BUSY         equ    128

PRINTER_FAILURE   equ   TIME_OUT+IO_ERR+NO_PAPER


    ;l'appel en C est : (void) reset_printer();
    ;aucune valeur retournée
procdef reset_printer
        mov ah,INIT_PRN
        mov dx,PRINTER_ID
        sti
        int PRINTER_INT
        xor ax,ax
        pret
pend reset_printer


    ;L'appel en C est : printer_busy();
    ;retourne 0 (FALSE) si disponible, une autre valeur (TRUE) si
occupé
procdef printer_busy
        mov ah,READ_STAT
        mov dx,PRINTER_ID
        sti
        int PRINTER_INT
        and ah,BUSY
        jz  printer_is_busy
        xor ax,ax
        pret
    printer_is_busy:
        mov ax,1
        pret
pend printer_busy



    ;  L'appel en C est :  (char) fetch_char(segm, offs);
    ;    Lit le caractère à segm:offs
    ;    et le retourne

procdef fetch_char,<<segm,word>,<offs,word>>
        push ds
        mov bx,offs
        mov ds,segm
        mov al,byte ptr ds:[bx]
        and ax,0ffH
```

```
        pop ds
        pret
pend fetch_char


    ;L'appel en C est : char_2_prn(outch);
    ; Ecrit un caractère sur PRINTER_ID
    ; retourne 0 si ok, ou code d'erreur

procdef char_2_prn,<<outch,byte>>
        mov al,outch
        mov ah,PRINT_CHAR
        mov dx,PRINTER_ID
        sti
        int PRINTER_INT
        test ah,PRINTER_FAILURE
        jnz printer_has_failed
        xor ax,ax
        pret
    printer_has_failed:
        mov al,ah               ;copie code erreur dans al
        and ax,PRINTER_FAILURE
        pret
pend char_2_prn


finish
```

VII - La Gestion mémoire sous DOS

LA GESTION MEMOIRE SOUS MS-DOS

[LISTING UN]

```
/*
  ==================  MCB.C  ======================================
    Ce programme parcours la liste des blocs de contrôle de
    mémoire (MCB) du DOS, calcule et affiche des informations
    relatives à chacun.
    R.J. Moore (C) 06 June 1988 Version 1.2. Peut être employé
    librement pour des utilisation non commerciales seulement.
    Compilé avec Turbo C, Version 1.5, en utilisant le modèle small,
    ce qui explique l'utilisation de pointeur huge et far.
    Testé sur IBM PC, IBM XT, IBM AT, clone TP/286 AT
    sous PC-DOS 2.1, 3.1, 3.2, et 3.3.  Tourne aussi sous
    MS-DOS 2.11 sur un DEC Rainbow.  PC-DOS 3.3 demande quelques
    modifications, comme indiqué dans les commentaires.
    ================================================================
*/

#include <stdio.h>
#include <dos.h>

/*----Declarations globales--------------------*/
struct MCB
{
     char chain;        /* 'Z' pour dernier MCB, 'M' pour les autres */
     unsigned  pid;     /* segment PSP propriétaire du MCB           */
     unsigned  psize;   /* Nb de paragraphes mémoire du MB attaché   */
     char unused [11]; /* Derniers 11 octets du MCB (réservés)       */
};

typedef struct MCB huge *PTRMCB;     /*PTRMCB est déclaré comme type
                                       de pointeur huge sur un MCB   */

/*----Prototypes de fonctions------------------*/
void main (void);
void far *ffmcb (void);    /* Retourne pointeur far sur le 1er MCB   */
void prn_header (void);    /* Imprime l'en-tête de table.            */
void prn_mcb (PTRMCB pm); /* Imprime MCB et informations associées. */
                           /* Imprime nom du propriétaire.           */
void prn_pid_own (unsigned pid,unsigned parent);

/*----main()-----------------------------------*/
/* Programme principal. Recherche l'adresse du 1er MCB, et parcours
   la chaîne des MCB en les affichant jusqu'à la fin de la chaîne.  */

void main()
{
  PTRMCB ptrmcb;              /* ptrmcb est un pointeur huge sur MCB */

   /* Obtient un pointeur sur le 1er MCB. Notez que ffmcb() retourne
      un pointeur FAR, qui est ensuite convertit en pointeur HUGE.
      Un pointeur FAR est suffisant  pour trouver le 1er MCB, car
      un tel pointeur peut adresser toute la mémoire. Mais ce pointeur
      doit être convertit en HUGE dans ptrcmb, car un MCB peut
      s'étendre sur plus de 64K, ce qui est la limite admise pour
      un pointeur FAR, car le segment d'un pointeur FAR ne peut
      changer. Ceci a évidemment été dur à trouver. Ces déclarations
      un peu spéciales peuvent être évitées si l'on utilise le
      modèle HUGE, mais cette méthode est en quelque sorte
      plus instructive.                                            */

  ptrmcb = (PTRMCB) ffmcb(); /* Obtient un pointeur FAR sur le 1er
                                MCB et le transforme en HUGE       */
  prn_header ();             /* Imprime l'en-tête de table         */
  prn_mcb(ptrmcb);           /* Imprime infos du 1er MCB           */

   /* Imprime infos pour tous les MCB suivants                     */
  do
  {
    ptrmcb += ptrmcb->psize + 1;   /* Pointeur sur MCB  suivant     */

    /* Chaque unité ajoutée à ptrmcb correspond à un paragraphe.
       Ajouter ptrmcb->taille incrémente donc le pointeur de la
       taille du MB contrôlé par le MCB. Le (+1) correspond au
       paragraphe du MCB.                                          */

    prn_mcb(ptrmcb);               /* Imprime infos MCB           */
  } while (ptrmcb->chain == 'M');  /* tant que non fin de chaine  */
                                   /* Imprime fin de table        */
  printf ("=============================================================");
  puts   ("===========");
}

/*----ffmcb()----------------------------------*/
/* Retourne un pointeur FAR sur le premier MCB en mémoire.
   Une déclaration explicite en pointeur FAR est nécessaire
   à cause de la compilation en modèle small.                      */

void far *ffmcb(void)
{
  union REGS regs;         /* REGS et SREGS definis dans dos.h.     */
  struct SREGS sregs;
  unsigned far *segmptr;  /*  Pointeur sur adresse de segment du MCB*/
  regs.h.ah=0x52;          /*  Fonction DOS non documenté no 0x52   */
  intdosx(&regs, &regs, &sregs);  /* ES:BX-2 pointe sur l'adresse de
                                     segment du 1er MCB en retour,
                                     et est copié dans segmptr     */
  segmptr=MK_FP(sregs.es,regs.x.bx-2);
  return MK_FP(*segmptr,0);        /* Retourne le pointeur lui-même. */
}                                  /* Le segment est pointé par     */
                                   /* segmptr. L'offset est à zéro  */
                                   /* (frontière de paragraphe)     */

/*-------prn_header()--------------------------*/
/* Imprime un en-tête décrivant les informations affichées
   sur les MCB par la fonction prn_mcb().
*/
void prn_header (void)
{
  printf ("================================================");
  puts   ("===============");
  puts   ("MCB MCB  ID PID      MB PAR- ENV  OWNER");
  puts   ("NO. SEG            SIZE ENT  BLK?");
  printf ("================================================");
  puts   ("===============");

/*    MCB NO. = Numéro ordinal du MCB traité (1,2,...).
      MCB SEG = adresse du segment (hex) du MCB
      ID      = Id de chainage, 'Z' si dernier MCB, 'M' sinon
      PID     = Id du programme possédant le MCB = adresse du segment
                du PSP. (Un PSP commence toujours sur une frontière
                de segment).
      MB SIZE = taille du MB associé (affiché en décimal)
      PARENT  = adresse du segment du parent du processus PID.
      ENV BLK?= 'O' si le MCB est un bloc d'environnement,
                'N' sinon.
      OWNER   = nom du programme associé au PID.
*/
}

/*----prn_mcb()--------------------------------*/
/* Imprime les infos sur un MCB dont l'adresse est passée en
   paramètre
*/

void prn_mcb (PTRMCB pm)
{
  static cnt = 0;          /* Compte le nb de fois que le parent est
                              égal au PID.                          */
  static mcbnum = 1;       /* No ordinal du MCB à imprimer          */
  unsigned parid;          /* Id parent (adresse du segment du
                              processus parent)                     */
  unsigned mcbseg;         /* adresse du segment du MCB             */
  char envf;               /* Contient 'O' ou 'N' selon que le MCB
                              est un bloc d'environnement ou non    */
  unsigned envseg;         /* Adresse de segment du bloc d'environ-
```

```
                                nement du PID                                */

  /* Adresse de l'id parent à l'adresse pid:16H                              */
  parid = * (unsigned far *) MK_FP (pm->pid,0x16);

  mcbseg = FP_SEG (pm);     /* Adresse du segment du  MCB                    */

  envseg = * (unsigned far *) MK_FP(pm->pid,0x2C);  /* Adresse             */
                              /* de segment de l'environnement               */
                              /* à l'adresse pid:2CH.                        */

    /* Si la valeur du segment du MCB plus un est égale à l'adresse
       de segment de l'environnement, alors le MCB contrôle un
       environnement                                                         */

  envf = mcbseg+1 ==  envseg ? 'Y' : 'N';

    /* Compte le nombre de fois que le parent et le pid sont
       identique. Si ce nombre est 1, le MCB appartient à
       COMMAND.COM                                                           */

  if (parid == pm->pid) cnt++;

    /* La détermination d'un bloc d'environnement ci-dessus est
       n'est pas terminée pour les versions du DOS de 2.0 à 3.2.
       Le procédé ci-dessus n'idetifiera pas le bloc d'environnement
       maitre possédé par la copie maitre de COMMAND.COM car la
       valeur de pid:2CH contient zéro, et non l'adresse de
       l'environnement maitre. Le procédé ci-dessous utilise le
       fait que l'environnement maitre suit le COMMAND.COM maitre
       en mémoire, alors que les environnements des autres
       programmes sont AVANT le programme lui-même en mémoire.
       A partir de DOS 3.3, pid:2CH pointe toujours vers
       l'environnement, même pour COMMAND.COM, et donc les lignes
       qui suivent sont inutiles (mais il n'est pas indispensable
       de les supprimer)                                                     */

  if (!envseg && cnt == 2) envf = 'Y';

    /* Imprime les infos du MCB, sauf le nom du propriétaire                */

  printf("%2.2u%06.4X%2.1c%06.4X%7lu%5.4X %-5.1c",
   mcbnum++,mcbseg,pm->chain,pm->pid,(long) pm->psize*16,parid,envf);

/* Appelle prn_pid_own() pour trouver et imprimer le nom du PID             */

  prn_pid_own(pm->pid,parid);
}

/*---prn_pid_own()-----------------------------*/
/* Imprime le nom du programme propriétaire en fonction du PID.
   L'adresse du parent sert à déterminer s'il s'agit du COMMAND.COM
   (vrai si PID=parent).
   Cette fonction utilise aussi le fait que certaines valeurs de
   PID sont réservées :

   pid = 0 signifie que le MCB est un bloc de mémoire libre
   pid = 8 signifie que le MCB est attribué à IBMDOS.COM/MSDOS.SYS
   pid = parent signifie que le MCB est attribué à COMMAND.COM (le seul
         programme qui soit son propre parent.

   Dans ces trois cas, le nom du propriétaire n'est pas lu dans
   le bloc d'environnement, car il ne s'y trouve pas.
   Les autres noms de propriétaires sont constitués du chemin d'accès
   complet, et ne sont disponibles qu'à partir de DOS 3.x.
*/

void prn_pid_own (unsigned pid,unsigned parent)
{
  unsigned far *envsegptr;  /* Ptr adresse seg de l'environnement   */
  char far *envptr;         /* Ptr environnement du PID             */
  unsigned far *envsizeptr; /* Ptr sur le mot envsize ci-dessous    */
  unsigned envsize;         /* Taille de l'environnement du PID     */

  /* Compteur ordinal du nombre de copies de COMMAND.COM en mémoire.
     (ccnum = 1 pour copie maitre, 2 pour 2e copie, etc...          */

  static unsigned char ccnum = 0;

  /* Valeur du PID lors de l'appel précédent */

  static prev_pid = 0xFFFF;

  switch (pid)
  {
      /* Nom particuliers pour deux valeurs du PID                  */
    case 0 : puts ("BLOC MEMOIRE LIBRE");return;
    case 8 : puts ("IBMDOS.COM/MSDOS.SYS");return;
  }

  /* pid:2CH = ptr sur adresse du segment de l'environnement du PID */
  envsegptr = (unsigned far *) MK_FP (pid,0x2C);

  /* Obtient le pointeur sur l'environnement lui-même               */
  envptr = (char far *) MK_FP (*envsegptr,0);

  /* Définit un pointeur qui contient la taille du bloc
     d'environnement. Doit pointer un paragraphe en arrière
     (là où commence le MCB d'environnement), et trois octets
     plus loin (là où se trouve le champ Taille dans le MCB)        */
  envsizeptr = (unsigned  far *) MK_FP(*envsegptr-1,0x3);

  /* Lit la taille de l'environnement en utilisant le pointeur
     ci-dessus, et convertit le nb de paragraphes en
     octets                                                         */
  envsize = *envsizeptr*16;


  if (pid == parent) /* c'est un command.com ? */
  {
    /* Si le PID précédent est différent du PID courant, on
       a trouvé une nouvelle copie de COMMAND.COM.
       ccnum compte le nombre de copies                             */

    if (prev_pid != pid) ccnum++;
    printf ("COMMAND.COM COPIE #%-2u\n",ccnum);

    prev_pid = pid;         /* Sauve PID courant- sera utilisé dans */
    return;                 /* un prochain appel à cette fonction   */
  }

  /* Boucle jusqu'à la fin de l'environnement                       */

  while (envsize)
  {
      /* envsize = nb d'octets restant à parcourir dans
         l'environnement. Le bloc est parcouru jusqu'à ce
         que le compteur soit nul, ou que l'on ait rencontre
         un NULL
      */
    while (--envsize && *envptr++);

      /* Teste si deux NULL se suivent, puis un 0001 après les
         deux NULL */

    if (!*envptr && *(unsigned far *) (envptr+1) == 0x1)
    {
       envptr +=3;
       break;                 /* chaine trouvée                       */
    }
  }

  if (envsize)
  {
     /* Imprime le nom s'il a été trouvé                               */
    while(*envptr) putchar(*envptr++);
    putchar('\n');
  }
  else
     /* Si le nom n'est pas trouvé
        (ne doit arriver qu'avec MS-DOS 2.x
     */
    puts ("PROPRIETAIRE INCONNU");
}
```

VIII - Un Gestionnaire de mémoire cache en C

Un gestionnaire de mémoire cache en C

[LISTING UN]

```
/* CACHE.H fichier en-tête */

typedef struct CACHE {
  int     recl;             /* longueur d'enregistrement */
  int     maxr;             /* nombre d'enregistrements dans le cache
*/
  long    hits;             /* nombre de succès pour cacfind() */
  long    miss;             /* nombre d'échecs pour cacfind() */
  long    adds;             /* nombre d'appels à cacnew() */
  int     (*proc)();        /* pointeur vers la fonction de traitement
*/
  long    idnt;             /* identificateur passé à proc() */
  long    *nums;            /* nombre d'enregistrements */
```

```
  char    *recs;              /* pointeur vers le premier enregistrement
*/
  short   *next;              /* tableau de pointeurs suivants */
  short   *prio;              /* tableau de pointeurs précédents */
  char    *mark;              /* tableau des marqueurs d'enregistrement à
traiter */
  short   lru;                /* plus vieil index */
  short mru;                  /* pls récent index */
  } CACDS;

long      cacamem;  /* taille mémoire allouée */

CACDS     *cacallo(int, int, int ( *)(), long);
char      *cacmem (int);
char      *cacnum (struct CACHE  *, long);
char      *cacold (struct CACHE  *);
int        cacflsh(struct CACHE  *);
int        cacnew (struct CACHE  *, short);
char      *cacfind(struct CACHE  *, long);
int        cacproc(struct CACHE  *, long);
int        cacunpc(struct CACHE  *, long);
int        cacstat(struct CACHE  *, long  *, long  *, long  *);
int        cacfree(struct CACHE  *);
```

[LISTING DEUX]

```
/* CACHE.C - Cache mémoire

  alan deikman 12/86

  Ces routines gèrent une mémoire cache d'enregistrements
  de taille fixe. Toute la mémoire est allouée avec la fonction
  statndard malloc().

  Toutes les fonctions, sauf cacallo(), ont comme premier paramètre
  un pointeur qui est retourné lors du premier appel de la
  fonction cacallo().

  cacallo()         Alloue le cache
  cacold()          Récupère le plus vieil enregistrement
  cacnum()          Numérote le plus vieil enregistrement et en fait le
plus récent
  cacflsh()         Gère les enregistrement marqués
  cacfind()         Trouve l'enregistrement n, et en fait le plus récent
  cacproc()         Marque un enregistrement "à gérer" quand il est
libéré
  cacunpc()         Supprime une marque sur un enregistrement
  cacstat()         Affiche les statistiques du cache
  cacfree()         Libère le cache

  */

#include <stdio.h>
#include <malloc.h>
#include "cache.h"

/* cacallo() - Alloue le cache */

CACDS *cacallo(num, recl, extf, idnt)
int       num;              /* nombre d'enregistrements à allouer */
int       recl;             /* longueur d'un enregistrement */
int       (*extf)();        /* pointeur sur la fonction externe de
traitement */
long      idnt;             /* paramètre à passer à la fonction externe
*/
{
  CACDS   *cac;
  char    *cacmem();
  int     i = 0;

  /* Initialise la structure cac avec les paramètres */

  if (num < 2) num = 2;
  cac = (CACDS *) cacmem(sizeof(CACDS));
  cac->recl = recl;
  cac->maxr = num;
  cac->proc = extf;
  cac->idnt = idnt;
  cac->hits =
  cac->miss =
  cac->adds = 0L;
  cac->nums = (long *)  cacmem(num * sizeof(long));
  cac->next = (short *) cacmem(num * sizeof(short));
  cac->prio = (short *) cacmem(num * sizeof(short));
  cac->mark = cacmem(num);
  cac->recs = cacmem(num * recl);

  /* initialise la chaîne lru/mru */

  while (i < num) {
    cac->next[i] = i + 1;
    cac->prio[i] = i - 1;
    cac->nums[i] = -1;
    cac->mark[i] = 0;
    i++; }

  cac->next[num - 1] = cac->prio[0] = -1;
  cac->mru = 0;
  cac->lru = num - 1;

  /* retourne le pointeur sur le cache */

  return cac; }


/* alloue la mémoire, avec test des erreurs */

char *cacmem(siz)
int       siz;
{
  char *c;
  c = malloc(siz);
  if (c == (char *) 0) {
    fprintf(stderr, "cacallo:  Allocation mémoire impossible.\n");
    fprintf(stderr, "Tentative d'allouer %d octets en plus des %ld déjà
alloués\n",
            siz, cacamem);
    exit(1); }
  cacamem += siz;
  return c; }

/* Numérote le plus ancien enregistrement, et en fait le plus récent.
   Si l'enregistrement était marqué pour traitement, appelle la
   fonction de traitement. Retourne un pointeur sur l'enregistrement */

char *cacnum(cac, num)
CACDS     *cac;             /* en-tête du cache */
long      num;              /* nouveau numéro du MRU */
{
  char    *rec = cac->recs + (cac->lru * cac->recl);

  /* appelle la fonction externe */

  if (cac->mark[cac->lru] && cac->proc)
    (*(cac->proc))(cac->idnt, cac->nums[cac->lru], rec);

  /* supprime la marque, et en fait le MRU */

  cac->mark[cac->lru] = 0;
  cac->adds++;
  cac->nums[cac->lru] = num;
  cacnew(cac, cac->lru);

  /* retourne l'enregistrement, prêt à l'usage */

  return rec; }


/* Retourne un pointeur sur l'enregistrement le plus ancien */
/* sans le modifier */

char *cacold(cac)
CACDS     *cac;             /* en-tête du cache */
{
  return cac->recs + (cac->lru * cac->recl); }


/* si une routine de traitement externe a été définie,
   traite tous les enregistrements marqués. */

cacflsh(cac)
CACDS     *cac;             /* en-tête du cache */
{
  int     i;
  char    *rec;

  if (!(cac->proc)) return;

  for (i = 0; i < cac->maxr; i++) if (cac->mark[i]) {
    rec = cac->recs + (i * cac->recl);
    (*(cac->proc))(cac->idnt, cac->nums[i], rec);
    cac->mark[i] = 0; }

  return; }


/* rend un enregistrement MRU */

cacnew(cac, rec)
CACDS     *cac;             /* en-tête du cache */
short     rec;              /* enregistrement à rendre MRU */
{

  /* si c'est déjà le mru, alors retour */

  if (rec == cac->mru) return;

  /* change le suivant du précédent */

  cac->next[cac->prio[rec]] = cac->next[rec];
```

```
  /* change le précédent du suivant
     s'il n'y a pas de suivant, c'est que c'est le lru.
     change le lru. */

  if (cac->next[rec] != -1)
    cac->prio[cac->next[rec]] = cac->prio[rec];
  else {
    if (rec != cac->lru) {
      fprintf(stderr, "cacnew: panique\n");
      exit(1); }
    cac->lru = cac->prio[rec]; }

  /* l'enregistrement est hors de la chaine.
     le rechaîne à la fin de la chaîne mru */

  cac->prio[cac->mru] = rec;
  cac->next[rec]      = cac->mru;
  cac->prio[rec]      = -1;
  cac->mru            = rec;

  /* terminé */

  return; }

/* cherche un enregistrement, et le rend mru */

char *cacfind(cac, num)
CACDS     *cac;              /* en-tête du cache */
long      num;               /* numéro de l'enregistrement à rechercher
*/
{
  int i;

  for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
    cac->hits++;
    cacnew(cac, i);
    return cac->recs + (i * cac->recl); }

  cac->miss++;
  return (char *) 0; }


/* marque un enregistrement à traiter */

cacproc(cac, num)
CACDS     *cac;              /* en-tête du cache */
long      num;               /* enregistrement à marquer */
{
  int i;

  for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
    cac->mark[i] = 1;
    return; }

  return; }

/* supprime la marque de traitement d'un enregistrement */

cacunpc(cac, num)
CACDS     *cac;              /* en-tête du cache */
long      num;               /* enregistrement à traiter */
{
  int i;

  for (i = 0; i < cac->maxr; i++) if (cac->nums[i] == num) {
    cac->mark[i] = 0;
    return; }

  return; }

/* affiche les statistiques du cache */

cacstat(cac, hit, mis, add)
CACDS     *cac;              /* en-tête du cache */
long      *hit, *mis, *add; /* valeurs à retourner */
{
  *hit = cac->hits;
  *mis = cac->miss;
  *add = cac->adds;
  return; }

/* libère le cache */

cacfree(cac)
CACDS     *cac;              /* en-tête du cache */
{
  cacflsh(cac);
  cacamem -= sizeof(CACDS) + (cac->maxr * sizeof(long)) +
             (cac->maxr * 2 * sizeof(short)) + (cac->maxr) +
             (cac->maxr * cac->recl);
  free(cac->recs);
  free(cac->prio);
  free(cac->next);
  free(cac->nums);
  free(cac);

  return; }
```

[LISTING TROIS]

```
#include <stdio.h>

#include "cache.h"

CACDS *cache;

main() {

  printf("Routine de test du cache\n");
  cache = cacallo(8, 128, (char *) 0, 1L);
  printf("Mémoire allouée = %ld\n\n",  cacamem);

  while (ctest()) cprint();

  exit(0); }


int ctest() {

  int opt, rec;
  long num;

  printf("\n1=old, 2=num, 3=find, 4=proc: ");
  scanf("%d", &opt);

  switch (opt) {
    case 0: return 0;
    case 1: printf("cacold retourne %lx\n", cacold(cache));  return 1;
    case 2: printf("entrez un enregistrement: ");
              scanf("%ld", &num);
              printf("cacnum retourne %lx\n", cacnum(cache, num));
              return 1;
    case 3: printf("entrez l'enregistrement à rechercher: ");
            scanf("%ld", &num);
            printf("cacfind retourne %lx\n", cacfind(cache, num));
            return 1;
    case 4: printf("entrez l'enregistrement à traiter: ");
            scanf("%ld", &num);
            cacproc(cache, num);
            printf("cacproc appelée\n");
            return 1;
    otherwise: return 1; }
  return; }

cprint() {

  register int i;

  printf("Cache :  succès=%ld  échecs=%ld  ajouts=%ld\n",
          cache->hits, cache->miss, cache->adds);

  printf("Bloc  Nombres Suiv  Préc  Marq. LRU=%d  MRU=%d\n", cache-
>lru, cache->mru);
  printf("——— ——— ——— ——— ———\n");
  for (i = 0; i < cache->maxr; i++)
    printf("%5d %7ld %5d %5d %4d\n", i, cache->nums[i],
                                                cache->next[i],
                                                cache->prio[i],
                                                cache->mark[i]);
  return; }
```

IX- PROTECT

```
;   Module : DKPROTEC.ASM       Routines protection
;                               et vérification dk
;   Projet         : PROTECT    -> lien PROTECT.OBJ
;                               Formatage non stand
;   Assembleur     : TASM 1.0    et vérification
;   Modèle mémoire : SMALL       authenticité disk.
;   par Jean Baptiste COURTOIS (c) 1991
;   pour Debug Magazine
;

        .MODEL SMALL

        .DATA

EXTRN           _buf : WORD     ; buffer 512 octets
```

```
Drive           DB      0       ; lecteur A: par défaut
Track           DB      39
Head            DB      0
Sector          DB      1

; ------------------------------;
; description des tables : pour chaque secteur,        ;
;   octet 1 : piste à formater                          ;
;   octet 2 : face (0=avant, 1=arrière)                 ;
;   octet 3 : n° secteur                                ;
;   octet 4 : nb d'octets par secteur (0=128, 1=256,    ;
;                                      2=512, 3=1024)   ;
; ------------------------------;

Format_Table1   DB      27h, 00h, 70h, 01h   ; 1er sect
                DB      27h, 00h, 71h, 01h   ; 2
                DB      27h, 00h, 72h, 01h   ; 3
                DB      27h, 00h, 73h, 01h   ; 4
                DB      27h, 00h, 74h, 01h   ; 5
                DB      27h, 00h, 75h, 01h   ; 6
                DB      27h, 00h, 76h, 01h   ; 7
                DB      27h, 00h, 77h, 01h   ; 8
                DB      27h, 00h, 78h, 01h   ; 9

Format_Table2   DB      26h, 00h, 01h, 00h   ; 1er sect
                DB      26h, 00h, 02h, 00h   ; 2
                DB      26h, 00h, 03h, 00h   ; 3
                DB      26h, 00h, 04h, 00h   ; 4
                DB      26h, 00h, 05h, 00h   ; 5
                DB      26h, 00h, 06h, 00h   ; 6
                DB      26h, 00h, 07h, 00h   ; 7
                DB      26h, 00h, 08h, 00h   ; 8
                DB      26h, 00h, 09h, 00h   ; 9

Protect_Msg     DB      'PROTECT!  Disquette '
                DB      'protégée par '
                DB      'utilitaire anti-piratage'
                DB      72 DUP ('C')

                        ; Nouvelle base Disque
Disk_Base       DB      0CFh
                DB      002h
                DB      025h
Byte_Sec        DB      001h    ; taille sect
                DB      009h    ; (mod. au cours vérif)
                DB      02Ah
                DB      0FFh
                DB      050h
                DB      0F6h
                DB      020h
                DB      004h

Retry_Count     DB      03      ; compteur essais vérif.


        .DATA?                  ; données non initialisées

Adr_Base        LABEL DWORD
Base_Off        DW      ?
Base_Seg        DW      ?


        .CODE

        PUBLIC  _protect
        PUBLIC  _verif

        LOCALS                  ; autorise labels locaux

; ---------- PROTECTION ----------;
;                                                       ;
; appel : int protect(int drive,int last_track)         ;
;                                                       ;
; ------------------------------;

_protect PROC

        PUSH    BP
        MOV     BP,SP
        PUSH    SI                  ; préserver var.
        PUSH    DI                  ;  registres du C
        MOV     AX,DS
        MOV     ES,AX               ; ES=DS
        MOV     AX,[BP+4]           ; lecteur
        MOV     Drive,AL
        MOV     AX,[BP+6]           ; n° dern. piste
        MOV     Track,AL
        CMP     AL,4Fh              ; si c'est une dk
        JNZ     Is_360              ;  720K, modifier
        MOV     BX,OFFSET Format_Table1 ; table format
        CALL    Ajust_Table         ; pour piste 79
        DEC     AL                  ; et piste 78
        MOV     BX,OFFSET Format_Table2
        CALL    Ajust_Table
Is_360:
        CALL    Change_Base
        CALL    Reset_Disk
        JC      IO_Err
                                    ; formater dern.piste
        MOV     BX,OFFSET Format_Table1
        CALL    Format
        JC      IO_Err

        MOV     Sector,70h          ; écrire sect 112
        CALL    Write_Msg
        DEC     Track

        MOV     BX,OFFSET Disk_Base
        MOV     BYTE PTR [BX+3],0   ; modifier dans DTP
                                    ; octet 4 = lg sect.

                                    ; formater av-dern
        MOV     BX,OFFSET Format_Table2   ;     piste
        CALL    Format
        JC      IO_Err

        MOV     Sector,4            ; écrire sect 4
        CALL    Write_Msg
                                    ; restaurer DTP
        CALL    Restore_Base        ; d'origine
        CALL    Reset_Disk
        JC      IO_Err
        XOR     AX,AX               ; code retour OK = 0
        JMP SHORT @@Fin
                                    ; Erreur E/S
IO_Err:
        CALL    Restore_Base
        MOV     AX,0FFFFh           ; retour code err. -1

@@Fin:
        POP     DI
        POP     SI
        POP     BP
        RET

_protect ENDP


; ---------- VERIFICATION ----------;
;                                                       ;
; appel : int verif(int drive,int last_track)           ;
;                                                       ;
; ------------------------------;

_verif  PROC

        PUSH    BP
        MOV     BP,SP
        PUSH    SI
        PUSH    DI
        MOV     AX,DS
        MOV     ES,AX               ; ES=DS
        MOV     AX,[BP+4]           ; lecteur
        MOV     Drive,AL
        MOV     AX,[BP+6]           ; n° dern. piste
        MOV     Track,AL

        CALL    Change_Base
        CALL    Reset_Disk
Retry:
        MOV     Byte_Sec,0          ; 128 oct / sect
        MOV     Sector,04           ; lire secteur 4
        MOV     CH,Track
        DEC     CH
        CALL    Read
        JNC     Not_Original        ; suspicion de copie
        CMP     AH,02               ; code erreur 02
        JNE     Not_Original        ; sinon copie
        MOV     Sector,09
        CALL    Read                ; lire secteur 9
        JNC     Not_Original
        CMP     AH,04               ; code erreur 04
        JNE     Not_Original        ; sinon copie
        MOV     Byte_Sec,01         ; 256 oct / sect.
        INC     CH                  ; vérif. dern. piste
        MOV     Sector,70h          ;       secteur 112
        CALL    Read
        JC      Not_Original

                                    ; original ?
        MOV     SI,OFFSET _buf      ; DS:SI -> Buffer
        MOV     DI,OFFSET Protect_Msg ;ES:DI -> Test_Msg
        MOV     CX,4
        CLD
        REPE CMPSW                  ; compare 8 car.
        JNZ     Not_Original        ; si ZF=0 -> copie
Original:                           ; c'est la dk orig.
        MOV     AX,1                ; code retour OK = 1
        JMP SHORT @@Fin
Not_Original:                       ; est-ce une copie ?
        DEC     Retry_Count
        CMP     Retry_Count,0
```

```
        JE      Is_Copy
        JMP     Retry
Is_Copy:                            ; si copie,
        XOR     AX,AX               ; code retour 0
@@Fin:
        PUSH    AX
        CALL    Restore_Base
        CALL    Reset_Disk
        POP     AX
        POP     DI
        POP     SI
        POP     BP
        RET

_verif  ENDP

; ———————— AJUSTER PISTE DANS TABLE ———————;
; entrée -> AL = piste, BX = offset Table                ;
; reg. modifiés : SI,CX                                  ;
; ———————————————————————————;

Ajust_Table     PROC

        XOR     SI,SI
        MOV     CX,9
Bis:
        MOV     [BX+SI],AL
        ADD     SI,4
        LOOP    Bis
        RET

Ajust_Table     ENDP

; ——— MODIFIER TABLE DES PARAMETRES ———;
; entrée -> rien                                         ;
; reg. modifiés : AX,BX,DX                               ;
; ———————————————————————————;

Change_Base     PROC

        PUSH    ES
        MOV     AH,35h
        MOV     AL,1Eh              ; lire adr IT 1Eh
        INT     21h
        MOV     AX,ES
        MOV     Base_Seg,AX         ; sauver Seg:off
        MOV     Base_Off,BX
        MOV     DX,OFFSET Disk_Base ;DS:BX = nouv. table
        MOV     AX,251Eh            ; fixer nouvelle adr.
        INT     21h                 ;  de l'IT 1Eh
        POP     ES
        RET

Change_Base     ENDP

; ——— RESTAURER TABLE PARAMETRES ———;
; entrée -> rien                                         ;
; reg. modifiés : AX,DX                                  ;
; ———————————————————————————;

Restore_Base    PROC

        PUSH    DS
        LDS     DX,Adr_Base
        MOV     AX,251Eh            ; fixer adr table
        INT     21h                 ; d'origine
        POP     DS
        RET

Restore_Base    ENDP

; ———————— FORMATAGE DISQUETTE ————————;
; entrée -> ES:BX = offset Table paramètres              ;
; reg. modifiés : AX,CX,DX,Flags                         ;
; ———————————————————————————;

Format  PROC

        MOV     DL,Drive
        MOV     DH,Head
        MOV     CL,01               ; n° sect.
        MOV     CH,Track
        MOV     AL,09   ; nb de secteurs à formater
        MOV     AH,05
        INT     13h
        RET

Format  ENDP

; ——————— ECRITURE SUR UN SECTEUR ———————;
; entrée -> ES : seg de Protect_Msg                      ;
; reg. modifiés : AX,BX,CX,DX,Flags                      ;
; ———————————————————————————;

Write_Msg PROC

        MOV     AH,03
        MOV     AL,01               ; nb de sect. à écrire
        MOV     CL,Sector           ; n° secteur
        MOV     CH,Track            ; n° piste
        XOR     DH,DH               ; face 0
        MOV     DL,Drive            ; lecteur
        MOV     BX,OFFSET Protect_Msg
        INT     13h
        RET

Write_Msg ENDP

; ———————— LECTURE DISQUETTE ————————;
; entrée ->  CH = piste, ES = segment _buf               ;
; reg. modifiés : AX,BX,CL,DX,Flags                      ;
; ———————————————————————————;

Read    PROC

        MOV     DL,Drive
        MOV     DH,Head
        MOV     CL,Sector
        MOV     AL,01
        MOV     BX,OFFSET _buf
        MOV     AH,02
        INT     13h
        RET

Read    ENDP

; ———————— RESET DISQUE ————————;
; entrée -> rien                                         ;
; reg. modifiés : AH,DL,Flags                            ;
; ———————————————————————————;

Reset_Disk PROC

        XOR     DL,DL               ; éviter reset HD
        XOR     AH,AH
        INT     13h
        RET

Reset_Disk ENDP
        END

; ———————————————————————————;
;                                                        ;
; Assemblage :    TASM (ou MASM) /mx dkprotec            ;
;                                                        ;
; ———————————————————————————;

/*
    Module : PROTECT.C          Protection contre
                                la copie.
    Projet          : PROTECT
    Compilateur     : TC 2.0    lier avec
    Modèle mémoire : SMALL      DKPROTEC.OBJ
    Auteur :  J.-B. COURTOIS (c) 1991
              pour Debug Magazine
/*

#include <stdio.h>
#include <dos.h>

#define  ESC   0x1B        /* touche Escape  */

typedef unsigned int    word;
typedef unsigned char   byte;

typedef struct {
   word  oct_sect;      /* nb d'octets par secteur     */
   byte  sect_clust;    /* nb de secteurs par cluster  */
   word  sect_boot;     /* nb de sect. réservés boot   */
   byte  nb_FAT;        /* nb de FAT                   */
   word  taille_FAT;    /* taille FAT en secteurs      */
   word  sect_pist;     /* nb de secteurs par piste    */
   word  n_tetes;       /* nb de têtes (faces)         */
   word  taille_root;   /* nb sect. du répert. racine  */
   word  sect_res;      /* nb total sect réservés      */
   word  last_track;    /* n° dernière piste disque    */
   } sdisk;

extern int protect(int,int);
                        /* retourne 0 si succès,       */
                        /*         -1 si erreur E/S    */
extern int verif(int,int);
                        /* retourne 1 si disk original*/
                        /*          0 si copie         */

byte buf[512];              /* buffer lecture secteurs */
char *IOErr_msg =
      "\r\n*** Erreur E/S disquette ***\r\n";

/*————————————————————————*/
/* Fonction : type_disk                                */
/* détermine les caractéristiques d'une disquette      */
/* Valeur retournée :  0 = dk DD 360 K    1 = dk 720K */
/*                    -1 = autre                       */
```

```
/*-----------------------------*/
int   type_disk(int drive,sdisk *dk)
{
   byte ID;
   word ntracks;
   byte *bptr;
   if(!(absread(drive,1,0,buf) == 0))   /* lire sect  */
   {                                    /* de boot    */
     cputs(IOErr_msg);
     exit(1);
   }
   bptr = buf + 11;
   dk->oct_sect = *(word *)bptr;      /* offset 11    */
   bptr += 2;                         /* du sect boot */
   dk->sect_clust = *bptr++;          /* off 13       */
   dk->sect_boot = *(word *)bptr;     /* off 14 (mot) */
   bptr += 2;
   dk->nb_FAT = *bptr++;                 /* off 16  */
   dk->taille_root = (*(word *)bptr)
          * 32 / dk->oct_sect;           /* off 17  */
   bptr += 2;
   ntracks = *(word *)bptr;              /* off 19  */
   bptr += 2;
   ID = *bptr++;                         /* off 21  */
   dk->taille_FAT = *(word *)bptr;    /* off 22 (mot) */
   bptr += 2;
   dk->sect_pist = *(word *)bptr;     /* off 24 (mot) */
   bptr += 2;
   dk->n_tetes = *(word *)bptr;       /* off 26 (mot) */
   dk->sect_res = dk->sect_boot +
     (dk->nb_FAT * dk->taille_FAT) + dk->taille_root;
   dk->last_track =                 /* n° dern. piste   */
     ((ntracks / dk->sect_pist) / 2 ) - 1;
   if(dk->sect_pist > 9) return(-1);
   /* indispensable car la marque 0F9h identifie     */
   /* à la fois les dk 720K et 1.2M...               */
   return((ID == 0xFD) ? 0 : ((ID == 0xF9) ? 1 : -1));
}

/*-----------------------------*/
/* Fonction : get_entree_FAT                        */
/* détermine le n° d'entrée de la FAT correspondant */
/* à un secteur logique donné                       */
/* Valeur retournée : n° entrée FAT                 */
/*-----------------------------*/
word  get_entree_FAT(int num_piste,int num_sect,
                     int num_face,sdisk *dk)
{
   word slog;
   slog =         /* déterminer n° secteur logique    */
     (dk->sect_pist * ((dk->n_tetes * num_piste)
      + num_face)) + num_sect - 1;
   return(((slog - dk->sect_res) / dk->sect_clust) + 2);
}

/*-----------------------------*/
/* Fonction : mark_bad                              */
/* marque comme défectueux les secteurs désignés    */
/* Valeur retournée : aucune                        */
/*-----------------------------*/
void mark_bad(byte *buffer,sdisk *dk,int piste,int face,
              int sect_deb,int nb_sect)
{
   int i;
   word e_fat;                      /* n° entrée FAT  */
   word x;
   word *w;
   for(i=sect_deb; i<(sect_deb+nb_sect); i++) {
     e_fat = get_entree_FAT(piste,i,face,dk);
     x = e_fat * 3 / 2;     /* entrées FAT sur 12 bits */
     w = (word *)(buffer + x);    /* w pointe sur mot  */
                              /* à l'adresse buffer+x */
     *w = (e_fat % 2)
         ? 0xFF70 + (*w & 0x000F)   /* entrée impaire */
         : 0x0FF7 + (*w & 0xF000);  /* entrée paire   */
   }
}

/*-----------------------------*/
/* Fonction : process_mark                          */
/* Lit la FAT puis la réécrit après marquage        */
/* Valeur retournée :  -1 = erreur lecture FAT      */
/*                      0 = succès                  */
/*                      1 = erreur écriture FAT 1   */
/*                      2 = erreur écriture FAT 2   */
/*-----------------------------*/
int   process_mark(int drv, sdisk *dk)
{
   byte *b;                   /* ptr sur tampon     */
   int ns = dk->taille_FAT;   /* nb de sect R/W     */
   int orig = dk->sect_boot;  /* sect de début R/W  */
   b = (byte *) malloc((ns * dk->oct_sect) + 1);
   if(!(absread(drv,ns,orig,b) == 0))   /* lire FAT  */
      return(-1);                       /* erreur lect */
   mark_bad(b,dk,dk->last_track-1,0,1,9);
                          /* marquer avant-dern. piste */
   mark_bad(b,dk,dk->last_track,0,1,9);
                              /* marquer dernière piste */
   if(!(abswrite(drv,ns,orig,b) == 0))  /* écr. FAT 1 */
      return(1);
   if(!(abswrite(drv,ns,orig+ns,b) == 0))/* écr FAT 2 */
      return(2);
   free(b);                        /* libère bloc alloué */
   return(NULL);
}
/*-----------------------------*/
/* Fonction : oui                                   */
/* enregistre réponse à question Oui/Non            */
/* Valeur retournée :  0 = Non    1 = Oui           */
/*-----------------------------*/
int   oui()
{
   char c;
   do c = (getch() & 0x5F);
   while((c != 'N') && (c != 'O'));
   return(c - 'N');
}

/*-----------------------------*/
/* Fonction : nb_disk()                             */
/* indique le nombre de lecteurs de disquettes      */
/* Valeur retournée :  nbre de lecteurs (1 ou 2)    */
/*-----------------------------*/
int   nb_disk()
{
   word equip;
   geninterrupt(0x11);
   equip = _AX;                  /* nb de lecteurs dans */
   return(((equip >> 6) & 0x03)+1); /* les bits 6,7   */
}


main(int argc, char *argv[])
{
  char   *init_msg =
      "PROTECT - Vérrouillage par soft et "
      "vérification de disquettes protégées.\r\n"
      "(c) 1987 A. Hoffman - 1991 J.B. Courtois\r\n\n";
  char  *choix_msg =
      "<P>rotection   -> format non standard de "
      "deux pistes.\r\n"
      "<V>érification -> vérifier l'authenticité "
      "d'une disquette.\r\n"
      "<A>bandonner   -> retour au DOS\r\n\n"
      "   Entrez votre choix (P/V/A) : ";
  char  *warn_msg =
      "\r\n\nCette disquette doit avoir été"
      " au préalable formatée normalement.\r\nAttention"
      " ! si cette disquette contient des données, "
      "certaines d'entre elles\r\npourraient être écra"
      "sées.\r\n\nConfirmez-vous l'opération (O/N) ? ";
  char   *start_msg =
      "\r\n\nAppuyer sur une touche pour démarrer la "
      "protection... ";
  char  *verif_msg =
      "\r\n\nAppuyer sur une touche pour démarrer la "
      "vérification... ";
  char  *OK_msg =
      "\r\n\nProtection effectuée sur les deux "
      "dernières pistes.\r\nVoulez-vous marquer les "
      "secteurs protégés comme défectueux (O/N) ? ";
  char  *mark_msg[] = {
      "\r\n\nSecteurs marqués.\r\n",          /* msg 0 */
      "\r\nErreur en écriture sur FAT 1. "    /* msg 1 */
            "Les secteurs n'ont pu être marqués.\r\n",
      "\r\nErreur en écriture sur FAT 2. "    /* msg 2 */
            "Les deux FAT ne concordent plus.\r\n"
            "Vous devriez reformater la disquette "
             "sous DOS.\r\n"
   };
  char  *orig_msg =
      "\r\n\nCette disquette a été protégée avec "
      "PROTECT.\r\n";
  char  *copy_msg =
      "\r\n\nCette disquette est une copie.\r\n";
  char  *Errdsk_msg =
      "\r\nErreur de disquette. Vous devez utiliser "
      "une disquette DD 360 Ko ou 720 Ko\r\n";
  char  *err_synt =
      "Vous devez préciser le lecteur. "
      " (ex. PROTECT B:)\r\n";
  char  *err_parm =
      "Paramètre invalide - Syntaxe : "
      "PROTECT A: ou PROTECT B:\r\n";
  char  *no_drive =
      "Vous avez spécifié un lecteur qui n'existe"
      " pas.\r\n";
  char  c;
  int    res;
  sdisk disk;
  int  drive = 0;              /* lect. A: par défaut  */
  int  n_dk;                   /* nbre de lecteurs     */

  clrscr();
```

```
   cputs(init_msg);
   n_dk = nb_disk();
   if(argc == 1) {
     if(n_dk > 1) {
       cputs(err_synt);
       exit(1);
     }
   }
   else {            /* vérif. validité ligne de comm. */
     if((argv[1][2]) || (argv[1][1] != ':')
        || ((drive = (argv[1][0] & 0x5F) - 'A') > 1)) {
         cputs(err_parm);
         exit(1);
     }
     else
       if((drive + 1) > n_dk) {
         cputs(no_drive);
         exit(1);
       }
   }
   cputs(choix_msg);                  /* afficher menu  */
   while(1) {
     c = (getch() & 0x5F);
     if((c=='A') || (c=='P') || (c=='V') || (c==ESC))
        break;
     putch(0x07);
   }
   switch(c) {
     case 'P' :                       /* option protection */
       cputs("\r\n\nIntroduire la disquette à protéger "
             "dans le lecteur ");
       putch(drive+'A');
       cputs(warn_msg);
       if(oui()) {
         cputs(start_msg);
         if (getch() != ESC) {
           if(type_disk(drive,&disk) < 0) {
             cputs(Errdsk_msg);
             break;
           }
          if(protect(drive,disk.last_track) == 0)
          {                           /* poser protection */
             cputs(OK_msg);
             if(oui()) {          /* effectuer marquage */
               if((res = process_mark(drive,&disk)) < 0)
                 cputs("\r\nErreur lecture disque.\r\n");
               else cputs(mark_msg[res]);
             }
          }       /* if(protect) */
          else cputs(IOErr_msg);
         }      /* if getch */
         else clrscr();
       }        /* if (oui) */
       else clrscr();
       break;

     case 'V' :                  /* option vérification  */
       cputs("\r\n\nIntroduire la disquette à vérifier "
             "dans le lecteur ");
       putch(drive+'A');
       cputs(verif_msg);
       getch();
       if(type_disk(drive,&disk) < 0) {
         cputs(Errdsk_msg);
         break;
       }
       if(verif(drive,disk.last_track) == 1)
          cputs(orig_msg);
       else cputs(copy_msg);
       break;

     default  :  clrscr();
   }                  /* de switch      */
}              /* de main  */
```

```
/*
    Module : PROTECT.C          Protection contre
                                la copie.
    Projet         : PROTECT
    Compilateur    : TC 2.0     lier avec
    Modèle mémoire : SMALL      DKPROTEC.OBJ
    Auteur :  J.-B. COURTOIS   */


#include <stdio.h>
#include <dos.h>

#define  ESC   0x1B        /* touche Escape  */

typedef unsigned int    word;
typedef unsigned char   byte;

typedef struct {
   word  oct_sect;     /* nb d'octets par secteur     */
   byte  sect_clust;   /* nb de secteurs par cluster  */
   word  sect_boot;    /* nb de sect. réservés boot   */
   byte  nb_FAT;       /* nb de FAT                   */
   word  taille_FAT;   /* taille FAT en secteurs      */
   word  sect_pist;    /* nb de secteurs par piste    */
   word  n_tetes;      /* nb de têtes (faces)         */
   word  taille_root;  /* nb sect. du répert. racine  */
   word  sect_res;     /* nb total sect réservés      */
   word  last_track;   /* n° dernière piste disque    */
   } sdisk;

extern int protect(int,int);
                       /* retourne 0 si succès,       */
                       /*         -1 si erreur E/S    */
extern int verif(int,int);
                       /* retourne 1 si disk original*/
                       /*           0 si copie        */

byte buf[512];              /* buffer lecture secteurs */
char *IOErr_msg =
      "\r\n*** Erreur E/S disquette ***\r\n";
/*-----------------------------*/
/* Fonction : type_disk                                */
/* détermine les caractéristiques d'une disquette      */
/* Valeur retournée :  0 = dk DD 360 K    1 = dk 720K */
/*                    -1 = autre                       */
/*-----------------------------*/
int   type_disk(int drive,sdisk *dk)
{
   byte ID;
   word ntracks;
   byte *bptr;
   if(!(absread(drive,1,0,buf) == 0))   /* lire sect  */
   {                                    /* de boot    */
     cputs(IOErr_msg);
     exit(1);
   }
   bptr = buf + 11;
   dk->oct_sect = *(word *)bptr;      /* offset 11    */
   bptr += 2;                         /* du sect boot */
   dk->sect_clust = *bptr++;          /* off 13       */
   dk->sect_boot = *(word *)bptr;     /* off 14 (mot) */
   bptr += 2;
   dk->nb_FAT = *bptr++;                   /* off 16  */
   dk->taille_root = (*(word *)bptr)
            * 32 / dk->oct_sect;           /* off 17  */
   bptr += 2;
   ntracks = *(word *)bptr;                /* off 19  */
   bptr += 2;
   ID = *bptr++;                           /* off 21  */
   dk->taille_FAT = *(word *)bptr;    /* off 22 (mot) */
   bptr += 2;
   dk->sect_pist = *(word *)bptr;     /* off 24 (mot) */
   bptr += 2;
   dk->n_tetes = *(word *)bptr;       /* off 26 (mot) */
   dk->sect_res = dk->sect_boot +
     (dk->nb_FAT * dk->taille_FAT) + dk->taille_root;
   dk->last_track =               /* n° dern. piste   */
      ((ntracks / dk->sect_pist) / 2 ) - 1;
   if(dk->sect_pist > 9) return(-1);
   /* indispensable car la marque 0F9h identifie      */
   /* à la fois les dk 720K et 1.2M...                */
   return((ID == 0xFD) ? 0 : ((ID == 0xF9) ? 1 : -1));
}
/*-----------------------------*/
/* Fonction : get_entree_FAT                           */
/* détermine le n° d'entrée de la FAT correspondant    */
/* à un secteur logique donné                          */
/* Valeur retournée : n° entrée FAT                    */
/*-----------------------------*/
word  get_entree_FAT(int num_piste,int num_sect,
                     int num_face,sdisk *dk)
{
   word slog;
   slog =          /* déterminer n° secteur logique    */
     (dk->sect_pist * ((dk->n_tetes * num_piste)
      + num_face)) + num_sect - 1;
   return(((slog - dk->sect_res) / dk->sect_clust) + 2);
}

/*-----------------------------*/
/* Fonction : mark_bad                                 */
/* marque comme défectueux les secteurs désignés       */
/* Valeur retournée : aucune                           */
/*-----------------------------*/
void mark_bad(byte *buffer,sdisk *dk,int piste,int face,
              int sect_deb,int nb_sect)
{
   int i;
   word e_fat;                        /* n° entrée FAT  */
   word x;
   word *w;
   for(i=sect_deb; i<(sect_deb+nb_sect); i++) {
     e_fat = get_entree_FAT(piste,i,face,dk);
     x = e_fat * 3 / 2;     /* entrées FAT sur 12 bits */
     w = (word *)(buffer + x);   /* w pointe sur mot   */
                                 /* à l'adresse buffer+x */
     *w = (e_fat % 2)
         ? 0xFF70 + (*w & 0x000F)   /* entrée impaire */
```

```
        : 0x0FF7 + (*w & 0xF000);  /* entrée paire   */
   }
}

/*-------------------------------*/
/* Fonction : process_mark                          */
/* Lit la FAT puis la réécrit après marquage        */
/* Valeur retournée :  -1 = erreur lecture FAT      */
/*                      0 = succès                  */
/*                      1 = erreur écriture FAT 1   */
/*                      2 = erreur écriture FAT 2   */
/*-------------------------------*/
int   process_mark(int drv, sdisk *dk)
{
   byte *b;                    /* ptr sur tampon       */
   int ns = dk->taille_FAT;    /* nb de sect R/W       */
   int orig = dk->sect_boot;   /* sect de début R/W    */
   b = (byte *) malloc((ns * dk->oct_sect) + 1);
   if(!(absread(drv,ns,orig,b) == 0))   /* lire FAT    */
      return(-1);                       /* erreur lect  */
   mark_bad(b,dk,dk->last_track-1,0,1,9);
                        /* marquer avant-dern. piste */
   mark_bad(b,dk,dk->last_track,0,1,9);
                           /* marquer dernière piste */
   if(!(abswrite(drv,ns,orig,b) == 0))  /* écr. FAT 1 */
      return(1);
   if(!(abswrite(drv,ns,orig+ns,b) == 0))/* écr FAT 2 */
      return(2);
   free(b);                         /* libère bloc alloué */
   return(NULL);
}

/*-------------------------------*/
/* Fonction : oui                                   */
/* enregistre réponse à question Oui/Non            */
/* Valeur retournée :  0 = Non    1 = Oui           */
/*-------------------------------*/
int   oui()
{
   char c;
   do c = (getch() & 0x5F);
   while((c != 'N') && (c != 'O'));
   return(c - 'N');
}

/*-------------------------------*/
/* Fonction : nb_disk()                             */
/* indique le nombre de lecteurs de disquettes      */
/* Valeur retournée :  nbre de lecteurs (1 ou 2)    */
/*-------------------------------*/
int   nb_disk()
{
   word equip;
   geninterrupt(0x11);
   equip = _AX;                  /* nb de lecteurs dans */
   return(((equip >> 6) & 0x03)+1); /* les bits 6,7   */
}


main(int argc, char *argv[])
{
  char   *init_msg =
      "PROTECT - Vérrouillage par soft et "
      "vérification de disquettes protégées.\r\n"
      "(c) 1987 A. Hoffman - 1991 J.B. Courtois\r\n\n";
  char  *choix_msg =
      "<P>rotection   -> format non standard de "
      "deux pistes.\r\n"
      "<V>érification -> vérifier l'authenticité "
      "d'une disquette.\r\n"
      "<A>bandonner   -> retour au DOS\r\n\n"
      "   Entrez votre choix (P/V/A) : ";
  char  *warn_msg =
      "\r\n\nCette disquette doit avoir été"
      " au préalable formatée normalement.\r\nAttention"
      " ! si cette disquette contient des données, "
      "certaines d'entre elles\r\npourraient être écra"
      "sées.\r\n\nConfirmez-vous l'opération (O/N) ? ";
  char   *start_msg =
      "\r\n\nAppuyer sur une touche pour démarrer la "
      "protection... ";
  char  *verif_msg =
      "\r\n\nAppuyer sur une touche pour démarrer la "
      "vérification... ";
  char  *OK_msg =
      "\r\n\nProtection effectuée sur les deux "
      "dernières pistes.\r\nVoulez-vous marquer les "
      "secteurs protégés comme défectueux (O/N) ? ";
  char  *mark_msg[] = {
      "\r\n\nSecteurs marqués.\r\n",          /* msg 0 */
      "\r\nErreur en écriture sur FAT 1. "    /* msg 1 */
            "Les secteurs n'ont pu être marqués.\r\n",
      "\r\nErreur en écriture sur FAT 2. "    /* msg 2 */
            "Les deux FAT ne concordent plus.\r\n"
            "Vous devriez reformater la disquette "
             "sous DOS.\r\n"
   };
  char  *orig_msg =
      "\r\n\nCette disquette a été protégée avec "
      "PROTECT.\r\n";
  char  *copy_msg =
      "\r\n\nCette disquette est une copie.\r\n";
  char  *Errdsk_msg =
      "\r\nErreur de disquette. Vous devez utiliser "
      "une disquette DD 360 Ko ou 720 Ko\r\n";
  char  *err_synt =
      "Vous devez préciser le lecteur. "
      " (ex. PROTECT B:)\r\n";
  char  *err_parm =
      "Paramètre invalide - Syntaxe : "
      "PROTECT A: ou PROTECT B:\r\n";
  char  *no_drive =
      "Vous avez spécifié un lecteur qui n'existe"
      " pas.\r\n";
  char  c;
  int    res;
  sdisk disk;
  int  drive = 0;              /* lect. A: par défaut  */
  int  n_dk;                   /* nbre de lecteurs     */
  clrscr();
  cputs(init_msg);
  n_dk = nb_disk();
  if(argc == 1) {
    if(n_dk > 1) {
      cputs(err_synt);
      exit(1);
    }
  }
  else {            /* vérif. validité ligne de comm. */
    if((argv[1][2]) || (argv[1][1] != ':')
       || ((drive = (argv[1][0] & 0x5F) - 'A') > 1)) {
         cputs(err_parm);
         exit(1);
    }
    else
      if((drive + 1) > n_dk) {
        cputs(no_drive);
        exit(1);
      }
  }
  cputs(choix_msg);                  /* afficher menu  */
  while(1) {
    c = (getch() & 0x5F);
    if((c=='A') || (c=='P') || (c=='V') || (c==ESC))
       break;
    putch(0x07);
  }
  switch(c) {
    case 'P' :                     /* option protection */
      cputs("\r\n\nIntroduire la disquette à protéger "
            "dans le lecteur ");
      putch(drive+'A');
      cputs(warn_msg);
      if(oui()) {
        cputs(start_msg);
        if (getch() != ESC) {
          if(type_disk(drive,&disk) < 0) {
            cputs(Errdsk_msg);
            break;
          }
         if(protect(drive,disk.last_track) == 0)
         {                           /* poser protection */
            cputs(OK_msg);
            if(oui()) {          /* effectuer marquage */
              if((res = process_mark(drive,&disk)) < 0)
                cputs("\r\nErreur lecture disque.\r\n");
              else cputs(mark_msg[res]);
            }
         }       /* if(protect) */
         else cputs(IOErr_msg);
        }      /* if getch */
        else clrscr();
      }        /* if (oui) */
      else clrscr();
      break;
   case 'V' :                   /* option vérification  */
      cputs("\r\n\nIntroduire la disquette à vérifier "
            "dans le lecteur ");
      putch(drive+'A');
      cputs(verif_msg);
      getch();
      if(type_disk(drive,&disk) < 0) {
        cputs(Errdsk_msg);
        break;
      }
      if(verif(drive,disk.last_track) == 1)
         cputs(orig_msg);
      else cputs(copy_msg);
      break;

    default  :  clrscr();
  }                  /* de switch      */
}              /* de main  */
```

Gestionnaires de périphériques MS-DOS en C

Une alternative au langage assembleur

Par Andy Klein
Traduction par Pierre Morel-Fourrier

Depuis la version 2.0, MS-DOS autorise l'ajout de gestionnaires de périphériques dans le but d'installer de nouveaux périphériques à votre système, sans qu'on soit obligé d'apporter des modifications au DOS lui-même. On spécifie les nouveaux périphériques à ajouter dans un fichier ASCII appelé CONFIG.SYS en utilisant la commande DEVICE=xxxxxxxx.yyy. Ces gestionnaires de périphériques permettent une indépendance du système vis-à-vis du matériel car le code spécifique au matériel est isolé dans le gestionnaire.

Cet article montre comment écrire les fonctions nécessaires à un gestionnaire de périphérique en C. Pour cela, nous décrirons le format d'un gestionnaire de périphérique, et comment en créer un en utilisant (a peu près) le code objet produit par un compilateur C. Certaines fonctionnalités de bas niveau sont spécifiques au compilateur et se rapportent à l'Aztec-C, mais la portabilité vers un autre compilateur C (ou même vers un autre langage de haut niveau) est possible. L'exemple choisi pour cet article (prndrv.asm, listing un) est un gestionnaire de périphérique simple qui implémente une imprimante parallèle en remplacement du gestionnaire standard PRN. Le choix s'est porté sur un gestionnaire simple pour pouvoir se concentrer sur l'écriture des gestionnaires en C, plutôt que de devoir se perdre en détails sur l'écriture d'un gestionnaire spécifique. Mais ce gestionnaire n'en est pas moins fonctionnel pour autant.

L'en-tête des requêtes

Le DOS communique avec ses gestionnaires de périphériques à l'aide d'un paquet appelé en-tête de requête. C'est une zone de mémoire système possédant un format défini. Cet en-tête contient le code de la commande que le gestionnaire doit exécuter, un mot d'état que le gestionnaire renseigne pour indiquer au DOS si l'opération a été réussie ou non, et la longueur du paquet. L'en-tête est suivi des données nécessaires à l'opération. Ces données varient en fonction de l'opération à effectuer. Pour notre exemple de gestionnaire, cette zone peut être considérée comme fixe, car elle contient le segment, l'offset, et le nombre d'octets à transmettre. Req_hdr est une structure C définie dans rh.h (listing deux). Elle est utilisée par les fonctions du gestionnaire pour accéder à l'en-tête. Une tâche importante à réaliser est de rendre accessible l'en-tête aux fonctions du gestionnaire (voir plus bas).

Quand le DOS appelle un gestionnaire pour lui faire effectuer une tâche, il procède en deux étapes. La première étape fait référence à la stratégie du gestionnaire et se passe quand un pointeur long (segment:offset) sur l'en-tête de requête est transmis au gestionnaire dans les registres ES:BX. La requête n'est pas exécutée à ce moment là, le gestionnaire sauve juste le pointeur long sur l'en-tête de requête. La deuxième étape est appelé interruption périphérique. Aucun paramètre n'est transmis, mais le pointeur long sur l'en-tête est récupéré, et l'opération demandée dans le code de commande de l'en-tête est exécutée. C'est la routine d'interruption qui effectue en fait le travail du périphérique, et elle doit aussi initialiser des données pour que l'on puisse appeler les fonctions C compilées. Le DOS appelle toujours la routine de stratégie, puis, immédiatement après, l'interruption. Il en découle qu'un gestionnaire de périphérique possède deux points d'entrée : les routines de stratégie et d'interruption. Ces points d'entrée sont appelés dev_strategy et dev_interrupt dans le listing prndrv.asm.

Format d'un gestionnaire

Les gestionnaires de périphériques doivent posséder un format particulier en vue d'être incorporés à MS-DOS. Ce format est différent d'un fichier EXE ou COM. Les gestionnaires doivent commencer par un en-tête de périphérique à l'offset 0 au début du fichier, et tout le segment logique du gestionnaire doit figurer dans un seul segment physique, comme dans un fichier COM. Le gestionnaire de périphérique est en effet simplement chargé en mémoire. Aucune relocation n'est effectuée, contrairement à un fichier EXE.

La principale difficulté que vous rencontrerez lorsque vous écrirez votre propre gestionnaire sera de le faire s'exécuter normalement lors de l'initia-

[SERVICE LECTEURS : INTERESSANT : 18 / MOYEN : 19 / MAUVAIS : 20]

lisation du système sans qu'il ne bloque votre PC. Une fois que vous avez dépassé ce stade, c'est du gâteau !

L'en-tête de périphérique

Le rôle de l'en-tête de périphérique est de fournir à MS-DOS les déplacements (offset) des points d'entrée des routines de stratégie et d'interruption du gestionnaire, et le nom du périphérique ou bien le nombre d'unités contrôlées par le gestionnaire. Les différents champs de l'en-tête sont :

- Pointeur vers l'en-tête suivant (4 octets)
- Attributs (2 octets)
- Pointeur vers la routine de stratégie (2 octets)
- Pointeur vers la routine d'interruption (2 octets)
- nom de l'unité ou nombre d'unités contrôlées (8 octets)

Le pointeur vers l'en-tête suivant est un champ de quatre octets que DOS utilise pour chaîner le gestionnaire suivant dans la chaîne des gestionnaires. Ces quatre octets doivent être initialisés à 0xFF dans prndrv.asm pour indiquer à MS-DOS qu'il n'existe qu'un seul gestionnaire dans le fichier.

A chaque bit du champ Attribut correspond un état, comme indiqué à la table 1. Notre gestionnaire fonctionne en mode caractère, et a donc le bit 15 à un. Les autres bits sont à zéro.

Bit	15	1=périphérique caractère	0=périphérique bloc
Bit	14	1=gère IOCTL	0=ne gère pas IOCTL
Bit	13	1=format non-IBM	0=format IBM (mode bloc)
		1=gère un état occupé	0=non (mode caractère)
Bit	12	0 (réservé)	
Bit	11	1=supporte media extractible	(mode bloc)
Bits	10	-5 (réservés) doit être 0	
Bit	3	1=unité horloge	0=pas unité horloge
Bit	2	1=unité NUL	0=pas unité NUL
bit	1	1=unité stdout	0=pas unité stdout
bit	0	1=unité stdin	0=pas unité stdin

Table 1 : Signification des bits du champ Attribut.

Le dernier champ (nom ou nombre d'unités) varie en fonction du type de gestionnaire. Si le gestionnaire fonctionne en mode caractère, alors ce champ contient le nom du périphérique sur huit caractères ASCII, justifié à gauche et complété par des espaces si nécessaire. Ce nom sera considéré par le DOS comme le nom du périphérique. Lorsque des gestionnaires en mode caractère sont chargés à l'initialisation, les périphériques installables sont reliés à la chaîne des périphériques, avant les périphériques par défaut. Quand le système effectue la recherche d'un nom de périphérique dans cette chaîne, il stoppe sur la première occurrence rencontrée. Si l'on utilise un nom de périphérique existant, pour le nouveau gestionnaire que l'on charge, ce dernier remplacera le périphérique existant. Et c'est exactement ce que fait notre gestionnaire : il remplace le gestionnaire standard PRN.

Notez qu'il existe une exception à cette règle. Le périphérique NUL ne peut pas être remplacé car il se trouve toujours en tête de la liste chaînée. Les nouveaux périphériques sont ajoutés après le périphérique NUL.

Les périphériques utilisables en mode bloc ne possèdent pas de nom. En revanche, DOS leur assigne des lettres dans l'ordre dans lequel ils sont chargés. Mais à la différence des périphériques qui fonctionnent en mode caractère, les périphériques en mode bloc par défaut sont toujours chargés en premier et ne peuvent jamais être remplacés.

Un périphérique en mode bloc peut contrôler plusieurs unités, et ce nombre d'unités est présent dans le dernier champ de huit caractères de l'en-tête de périphérique. Le premier caractère de ce champ représente le nombre d'unités, les autres caractères doivent être des espaces. Le nombre d'unités contrôlées peut être modifié ultérieurement lorsque la routine d'initialisation du périphérique est exécutée. Un exemple d'utilisation de plusieurs unités par un seul périphérique est quand deux disques durs sont reliés à un seul contrôleur.

Le programme d'en-tête du gestionnaire

Dans le but d'écrire les fonctions d'un gestionnaire de périphérique en C, un certain nombre de choses un peu spéciales doivent être faites. D'abord, avant qu'une fonction C compilée ne soit appelée, l'environnement (registres de segments et pointeur de pile) doit être modifié pour correspondre à celui que le code généré par le compilateur attend. Normalement, cette initialisation est effectuée par un programme d'en-tête fourni avec le compilateur. Pour le compilateur Aztec C en modèle small, cette routine s'appelle sbegin.asm. Toutes les fonctions C compilées par l'Aztec C font une référence externe à une fonction appelée $begin, forçant ainsi l'éditeur de liens à extraire la fonction $begin de la librairie standard c.lib, et à l'inclure dans le programme exécutable. $begin appelle la fonction Croot, qui appelle à son tour la fonction main. En temps normal, $begin initialise l'environnement et appelle Croot, qui à son tour, appelle main. Les autres compilateurs possèdent une procédure d'initialisation similaire.

Mais la plupart des initialisations qui sont effectuées ne doivent pas figurer dans l'en-tête d'un gestionnaire de périphérique, comme par exemple l'analyse de la chaîne argv, l'allocation du tas, l'accès aux variables d'environnement du DOS, ainsi que d'autres initialisations indispensables pour certaines fonctions de la librairie. Supprimer ces initialisations implique que vos fonctions C ne pourront pas appeler certaines fonctions de la bibliothèque C, notamment malloc(), free(), et les fonctions d'entrée/sortie. En outre, toutes les variables globales devront être explicitement initialisées. Une tâche importante effectuée lors de ces initialisations, et qui doit être incorporée dans le gestionnaire, est de créer une pile propre au programme.

Dans notre exemple de gestionnaire de périphérique, les tâches d'initialisation sont effectuées en remplaçant les

routines d'initialisation normales par la fonction d'interruption du périphérique. Cette fonction initialise les registres de segment de la manière attendue par les fonctions compilées. Il n'y a pas de fonction main() car le point d'entrée du programme est le label dev_interrupt dans prndrv.asm. Il est nécessaire de créer une fonction $begin pour éviter que l'éditeur de liens ne génère une erreur, même si cette fonction n'est jamais appelée. Si vous utilisez un autre compilateur, vous devrez remplacer ces fonctions par celles que votre compilateur réclame. Si le source des fonctions d'initialisation est fourni, les modifications en seront grandement facilitées. Notons que les fabricants de compilateurs qui ne fournissent pas le source des fonctions d'initialisation limitent l'usage de leur compilateur dans le cas d'une utilisation poussée. Vous pourrez vous rendre compte de ce que votre compilateur génère comme code pour l'initialisation des programmes en compilant quelques exemples simples et en examinant le code généré.

Les périphériques installables sont reliés à la chaîne des périphériques, avant les périphériques par défaut.

La fonction de stratégie du périphérique

La routine de stratégie est appelée avec paramètre : un pointeur long sur l'en-tête de requête dans la paire de registres ES:BX. La fonction de stratégie n'exécute pas la requête, elle se contente juste de sauvegarder le pointeur. Dans notre exemple, ce pointeur est stocké dans les variables req_hdr_seg pour le segment, et req_hdr_off pour l'offset.

La fonction d'interruption du périphérique

La routine d'interruption du périphérique exécute la requête décrite dans l'en-tête de requête. Bien que cette fonction soit appelée routine d'interruption, elle est invoquée par le DOS à l'aide d'un appel FAR, et pas par une interruption du processeur.

Cette fonction doit effectuer sept tâches cruciales, dont certaines correspondent à l'en-tête d'initialisation d'un programme classique. Le listing prndrv.asm possède ainsi sept sections annotées ETAPE 0, ETAPE 1, etc...

- L'étape 0 sauve l'état de la machine. Tous les registres du processeur sont empilés dans la pile DOS qui est supposée assez grande. Puis, les pointeurs de pile DOS (les registres ss et sp) sont sauvegardés dans les variables du segment de code caller_ss et caller_sp.

- L'étape 1 initialise le segment du périphérique. L'adresse de ce segment est stockée dans le registre ax. Notez que tous les segments logiques du gestionnaire sont dans un seul segment physique. Quand le gestionnaire est appelé, la valeur du registre cs correspond à la valeur de tous les segments.

- L'étape 2 rend l'en-tête de requête adressable. Le pointeur long sur l'en-tête a été sauvegardé lors de l'appel de la routine de stratégie. La fonction d'interruption utilise donc cette information pour recopier l'en-tête DOS dans un en-tête privé dans le segment local.

- L'étape 3 achève de rendre les données adressables. La fonction d'interruption copie l'adresse du segment du gestionnaire de ax dans les registres ds, es et ss. L'offset de la pile privée est rangé dans le registre sp. Enfin, le registre sp est recopié dans le registre bp pour que les fonctions C qui utilisent bp puissent être appelées.

- L'étape 4 appelle la première fonction C, qui à son tour en appellera d'autres pour que la requête soit effectuée. La fonction appelée est driver_ functions (cmd_code) qui possède un paramètre : le code de commande de la requête. L'appel d'une fonction C en assembleur implique d'empiler les paramètres sur la pile, d'appeler la fonction, puis de dépiler les paramètres au retour de la fonction.

- L'étape 5 met à jour l'en-tête de requête DOS. La requête a été exécutée. Un code d'état ainsi que des informations spécifiques au type de requête se trouvent dans la copie locale de l'en-tête de requête. Pour que le DOS puisse accéder à ces informations, l'en-tête de requête privé est recopié dans l'en-tête DOS.

- L'étape 6 restaure l'état antérieur à l'appel de la routine d'interruption. Le pointeur de pile DOS sauvegardé dans les variables caller_ss et caller_sp est restauré, les registres qui ont été empilés dans la pile DOS sont restaurés, et la routine d'interruption se termine, la tâche est terminée.

Implémenter un périphérique

Après avoir terminé le code d'en-tête DOS du gestionnaire de périphérique, attaquons nous aux fonctions qui implémentent le périphérique. Chaque périphérique possède des caractéristiques et des besoins différents, et l'implémentation dépend étroitement du type de périphérique.

Le périphérique présenté ici est extrêmement simple. Une imprimante reçoit des caractères, et seules des fonctions de sortie doivent être écrites. Le code d'en-tête du gestionnaire ne change pas en fonction du périphérique. Seules les fonctions d'implémentation varient.

La fonction driver_functions (cmd_code) est appelée par l'en-tête du gestionnaire de périphérique. Cette routine se trouve dans drvfunc.c (listing trois). Elle se comporte comme un automate qui appelle indirectement une fonction correspondant au code de commande passé en paramètre. Function_table est déclaré dans drvfunc.c comme un tableau de pointeurs de fonctions initialisé avec les adresses des fonctions que le gestionnaire de périphérique implémente. Les fonctions non implémentées pointent vers bad_cmd, une routine qui retourne un code d'erreur indiquant que la commande est invalide (voir le fichier error.c, listing quatre). Le code de commande représente un état qui détermine la fonction à appeler.

Si vous n'êtes pas familier avec les tableaux de pointeurs de fonction en C, en voici une brève description. Chaque élément du tableau est un pointeur (une adresse) sur une fonction, c'est-à-dire (en modèle small 8086) un offset dans le segment de code. L'index dans le tableau détermine la fonction à appeler, de la même manière qu'un index dans un tableau d'entier

détermine quel entier utiliser. Si pointeur_fonction représente l'adresse d'une fonction, on appelle la fonction correspondante en écrivant (*pointeur_fonction)(); et en étendant ce schéma à un tableau, on écrira :

```
(*tableau_de_pointeurs_de_fonction [index])();.
```

Notez que toutes les fonctions du tableau doivent posséder le même nombre de paramètres, sinon des effets indésirables pourraient se produire.

Il n'y a rien de vraiment ésotérique. Les programmeurs assembleur appellent cela une table de sauts. Cette construction produit un code très efficace, ce qui est bien ce que l'on recherche dans le cas d'un gestionnaire de périphérique.

Une alternative à l'emploi d'un tableau de pointeur serait l'utilisation d'une instruction switch, ou bien d'une liste d'instructions if... else qui seraient compilés en une longue série de tests et branchements.

Tous les gestionnaires de périphérique doivent posséder une fonction d'initialisation appelée une seule fois lorsque le gestionnaire est chargé. Dans notre exemple, elle s'appelle init() (listing cinq). La fonction d'initialisation d'un gestionnaire de périphérique en mode caractère doit retourner à MS-DOS un pointeur long (segment:offset) vers la fin du gestionnaire, et positionner le mot d'état. L'adresse de fin du gestionnaire est constitué de l'adresse du segment (obtenu par appel de la fonction show_cs, listing six) et d'un pointeur court (offset seulement) sur la variable entière _Uend. Cette variable est créée par l'éditeur de liens du compilateur Aztec, et insérée automatiquement à la fin du segment de données. Si vous utilisez un autre compilateur, vous devrez créer un nouveau module contenant une variable entière que vous appelerez _Uend, et vous devrez ajouter ce module après tous les autres lors de la phase d'édition de liens.

Les autres fonctions du gestionnaire permettent d'afficher un message sur l'écran, d'initialiser le port parallèle, de modifier la police de l'imprimante.

Pour un autre type de périphérique, l'initialisation serait différente. Si le périphérique était utilisable en mode bloc (et non caractère), l'initialisation devrait en plus modifier le nombre d'unités, et retourner un pointeur vers un tableau de BPB (BIOS parameter blocks, bloc de paramètres du BIOS).

En tant qu'imprimante, la fonction principale s'appelle output() (listing sept). Cette routine lit dans l'en-tête de requête le nombre d'octets à envoyer ainsi que l'adresse (segment:offset) où se trouvent les données à imprimer. Chaque caractère est envoyé un par un à l'imprimante à l'aide de la fonction char_2_prn() (listing huit). Si un code d'erreur est retourné par cette routine, la fonction error() est appelée. Quand la requête est terminée, le bit « terminé » du mot d'état est positionné.

Bien sûr, il s'agit ici d'un gestionnaire simple, mais le principe serait le même pour un cas plus complexe.

Le code de retour

Le gestionnaire retourne une information au système par l'intermédiaire du mot d'état de l'en-tête de requête. Il s'agit un entier non signé appelé statut dans la structure d'en-tête de requête req_hdr (voir fichier rh.h). Le bit 15 est le bit d'erreur. Il doit être positionné à un lorsqu'une erreur survient pendant l'exécution de la requête. Les huit bits de poids faible du mot d'état indiquent alors un numéro d'erreur. Les codes d'erreur sont gérés dans le fichier error.c, et dans notre cas, les seuls codes d'erreur possibles sont ceux qui sont retournés par le port parallèle BIOS.

Si aucune erreur ne survient, le bit huit est positionné à un, indiquant ainsi que la requête s'est déroulée correctement.

Tous les gestionnaires de périphériques doivent inclure une fonction d'initialisation.

L'édition de liens

La phase d'édition de liens d'un gestionnaire de périphérique est différente de celle d'un programme classique. Tous les segments logiques doivent en effet se trouver dans le même segment physique, et l'en-tête de périphérique doit se trouver à l'offset 0 du fichier. L'une des raisons pour lesquelles nous avons utilisé le compilateur Aztec est que l'éditeur de liens nous permet d'effectuer ce travail simplement.

La ligne de commande pour effectuer une édition de liens correcte avec le compilateur Aztec est :

```
ln -t -b 0 -c 0 -o prndrv.com prndrv.o
[liste des fichiers et des librairies]
```

où -t sauve la table des symboles dans le fichier prndrv.sym, -b 0 initialise l'adresse de base à zéro (identique à ORG 0), -c 0 fait commencer le segment de code à l'offset zéro dans le segment physique, et -o prndrv.com signifie que le fichier exécutable sera prndrv.com (la spécification d'un fichier COM implique que tous les segments logiques se trouveront dans le même segment physique).

Si vous utilisez un autre compilateur, vous devrez adapter ceci à votre environnement. Vous pouvez essayer d'utiliser une instruction GROUP dans le fichier prndrv.asm pour que les segments de donnée et de code soient confondus, et rappelez vous que vous devez utiliser un offset par rapport au groupe et pas par rapport au segment lorsque vous utiliserez l'opérateur OFFSET.

Andy Klein, 3801 N. 16th St. #222, Phoenix, AZ 85106.
La société d'Andy, Micro Quantitative Sciences, développe des interfaces et des gestionnaires de périphériques pour IBM-PC.

Gestionnaire de periphériques MS - DOS en C : copyright **Dr Dobb's Journal** M&T Publishing, Inc, Redwood City, CA USA 1990. All right reserved.

PODIUM

UNE PUCE QUI FAIT UN GRAND PAS EN AVANT

par *ART*

Les ordinateurs bâtis sur la base de nouveau processeur INTEL i486 sont arrivés. Plusieurs marques bien connues proposent des systèmes puissants, dont le prix en France varient autour de quelque 50.000 F. HT. Vous les connaissez tous : IBM, COMPAQ, ZENITH, AST, OLIVETTI ou TANDON. Connaissez-vous Polywell?

Dans cette nouvelle rubrique, nous allons vous présenter des matériels ou logiciels que nous avons particulièrement aimé, pour leur utilité, originalité ou qualité. Si vous appréciez particulièrement un produit, soumettez-nous sa candidature. Les bons produits méritent d' être connus!

Polywell Europe, filiale de deux sociétés américaines : Polywell Computers Inc. (micro-ordinateurs) et Supercom of America (périphériques) a été crée en octobre 1990, dans la banlieue nord de Paris, à Villepinte. En USA, Polywell appartient à cette catégorie mal connue en France, des (relativement) petits constructeurs qui assemblent des ordinateurs à partir des composants (cartes et sous-ensembles) disponibles sur le marché, ajoutant le savoir faire, qualité et compétence à un ou l'autre élément essentiel, de leur propre cru. On peut citer Northgate, Cheetah, Gateway 2000, ASTRIX ou - Polywell - les marques qui font bonheur des « initiés ». En fait, surtout en Californie du Nord, presque toutes les grandes boutiques ont leur propre « marque » d'ordinateurs. Assemblés à partir des sous-ensembles standard disponibles sur le marche, ces machines se distinguent surtout par le prix sensiblement inférieurs par rapport aux prix des machines « poids-lourds ». En France, Polywell se veut apporter un « service sur mesure » à des SSII et dévelop

[SERVICE LECTEURS : INTERESSANT = 36 / MOYEN = 37 / MAUVAIS = 38]

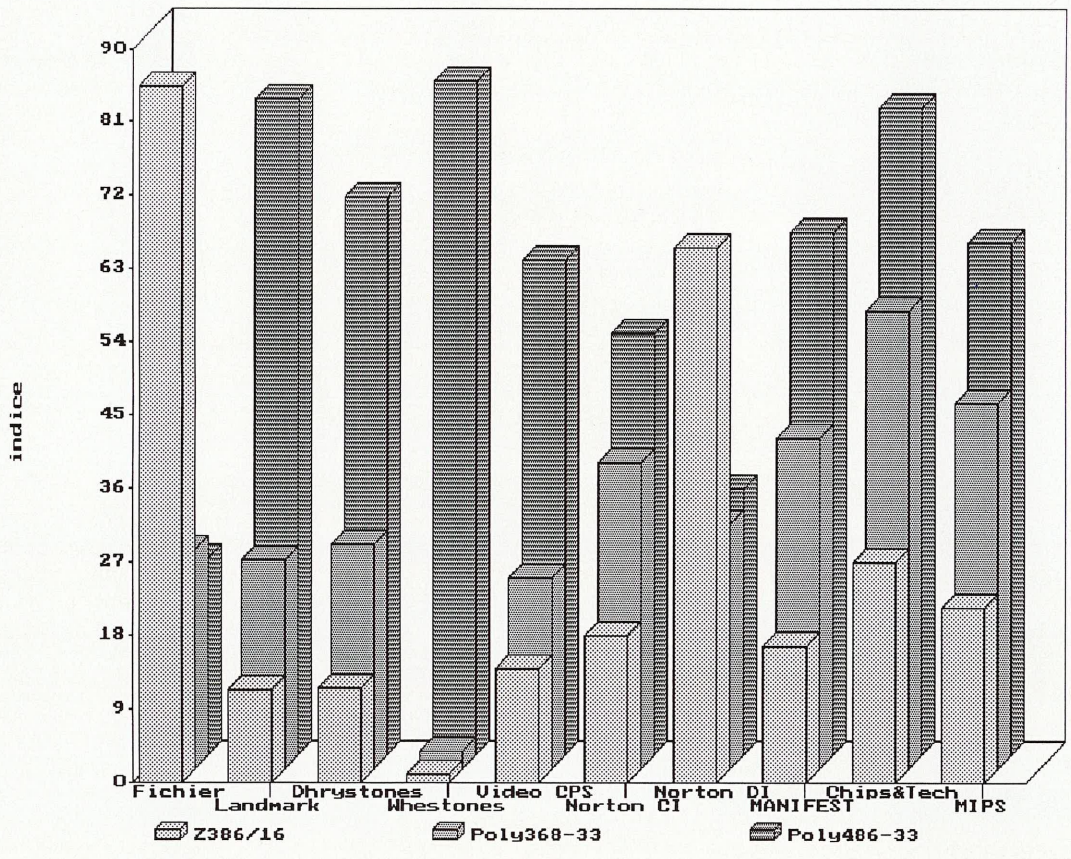
peurs professionnels, installateurs des systèmes « clef en main ». Ce qui distingue Polywell d'autres est la capacité d'assembler les systèmes « sur mesure », selon les spécifications du client. Polywell assure aussi une garantie gratuite : totale, d'un an pièces et main d'oeuvre, 2 ans pièces et main d'oeuvre sur la carte mère et 5 ans main d'oeuvre (pièces exclues), toutes en retour atelier.

« Revenons à nos moutons ». Nous avons eu envie de comparer deux machines le plus proches possible, dont la différence se situe seulement au niveau du processeur lui-même : un i386 contre un i486, même configuration, même vitesse. Polywell nous a prêté deux machines : un Poly486-33AS et un Poly386-33PF, fonctionnant tous les deux à 33 MHz, équipés d'une même carte vidéo (ProDesigner II avec 1Mo), même contrôleur des disques durs (OMTI 8620 ESDI à interlacement 1:1, caché 32Ko) et même disque dur NEC 150Mo à 16ms, bus AT (ISA). Pour avoir une référence plus « usuelle », nous avons aussi utilisé un « vieux » système à base d'un i386/16MHz (ZENITH Z386/16MHz avec 16Ko du cache externe), équipé d'un contrôleur classique SASI et d'un disque dur MFM à 28ms.

Pour les comparer, nous avons d'abord procédé à la préparation du système : formatage intégral des disques durs et l'installation d'un DR DOS 5.0 de Digital Research qui, à la différence d'un MS-DOS « classique », exploite les capacités des nouvelles machines. Ensuite, nous avons installé nos outils de test : Landmark, Diagsoft, Norton, Manifest de Quarterdeck, et Chips & Technologie. Un dernier test était destiné à mesurer la productivité « perceptible » de l'ordinateur : un large fichier de 25 Mo a été extensivement manipulé (remplacements, indexations multiples et 5 recopies) par un programme sous FoxPRO 1.0.

Les résultats sont donnés dans le tableau qui suit : comme on pouvait s'attendre, l'ordinateur à la base d'un i486 sort vainqueur, les meilleurs résultats se situent au niveau de la puissance de calcul - i486 écrase littéralement ses concurrents. Mais ce qui compte surtout pour l'utilisateur, c'est la productivité « sensible ». Si votre souci quotidien n'est pas le calcul mais manipulation des fichiers, un contrôleur caché performant va changer votre vie - regardez le tableau.

La machine Poly386-33PF est un « desktop »(qu'on peut également poser à la verticale - un pied est fourni), d'une construction assez classique. La carte mère est fabriquée en USA pour Polywell par OMNITEL en 8 couches. La capacité de la mémoire sur la carte mère peut atteindre 16Mo, par des SIMM's économiques de 256Ko, 1MO ou 4Mo. Un cache statique de 64Ko à 15ns externe permet de travailler avec des mémoires lentes à 70-80 ns, sans des états d'attente. Si vous avez besoin, un emplacement pour le coprocesseur 80387, Weitek ou Cyrix est prévu. Un BIOS Phenix (version A 386 1.00) permet les tests et la configuration interactifs sur l'écran. Deux connecteurs 8 bit et six connecteurs 16 bits acceptent les cartes au standard AT à 8MHz. Une alimentation (approuvé UL/CSA) délivre 220 Watts, permettant de connecter autant des disques et cartes dont vous avez besoin. Deux ports série et un parallèle sont apportés par une carte enfichée dans un des connecteurs 8 bit. Le système est livré avec un excellent clavier Cherry G81-1000, dont la qualité n'est plus à vanter. Le « bonus » plus appréciable est apporté par la carte vidéo Super-VGA, fabriquée par ORCHID - ProDesigner II, équipée d'un mégaoctet de la mémoire. A part d'une excellente vitesse d'affichage (connecteur 16 bits),



Suite de l'article page 73

Des arbres programmatiques pour une analyse naturelle...

par
Jean Christophe Riat

Comme l'écrivain à la recherche de l'inspiration devant sa feuille blanche, trop de programmeurs réfléchissent à leur application devant une console. Ils estiment, à tort, avancer plus rapidement en se dispensant de toute analyse fonctionnelle préalable. Les arbres programmatiques présentés dans cet article, constituent un formalisme graphique pour décomposer un problème de manière progressive. Il est fortement conseillé de les utiliser avant toute écriture de code...

[SERVICE LECTEURS : INTERESSANT = 21 / MOYEN = 22 / MAUVAIS = 23]

Si la phase d'analyse peut paraître fastidieuse au programmeur débutant, son intérêt n'est souvent plus à démontrer à son homologue plus expérimenté. C'est lors de cette première étude que le problème est clairement posé, et que les algorithmes pour le résoudre sont définis. Si l'analyse est menée à son terme, l'écriture du code se transforme en une simple traduction en langage informatique des opérations ainsi détaillées. Un document d'analyse constitue non seulement une excellente base pour la programmation, mais également un support de communication efficace. La description de la logique du programme simplifie la compréhension du code pour quelqu'un d'autre que le concepteur, voire pour le concepteur lui-même, après une période consacrée à d'autres activités que le suivi de son application...

Les organigrammes sont une méthode de description bien connue des informaticiens. Basés sur une représentation séquentielle du cheminement du pointeur lors de l'exécution du programme, ils sont bien adaptés pour les langages axés sur les branchements conditionnels comme le BASIC ou l'assembleur. Mais ils ne conviennent absolument pas pour des langages structurés comme Pascal, C ou ADA. Pour ces derniers existe un formalisme beaucoup moins répandu, les arbres programmatiques. Comme nous allons le voir, celui-ci permet d'une part d'affiner progressivement la définition du problème à traiter, et d'autre part de décrire la séquence des opérations nécessaires à sa résolution.

Le formalisme des arbres programmatiques...

La seule façon d'appréhender un problème compliqué est de le décomposer en sous-problèmes plus simples. Si ces derniers sont jugés encore trop complexes, ils seront à leur tour éclatés. Et ainsi de suite jusqu'à obtenir une décomposition suffisamment fine pour que chacun des sous-problèmes terminaux puissent être facilement traités. En adoptant une représentation graphique, on obtient un arbre hiérarchique d'analyse, comme celui de la figure 1. Ecrire un programme pour résoudre le problème initial revient alors à juxtaposer des morceaux de code qui traitent chacun une des boîtes terminales de l'arbre. Le séquencement ainsi obtenu correspond à une lecture de l'arbre de la gauche vers la droite.

Un programme n'est cependant que trés rarement constitué uniquement d'opérations séquentielles. Les langages structurés, précédemment évoqués, disposent de deux catégories d'instructions non séquentielles:

- les instructions d'alternative conditionnent l'exécution d'actions à la vérification de prédicat,

- les instructions de répétitivité permettent d'itérer l'exécution d'actions.

Le langage Pascal, qui sera pris comme référence dans la suite de cet article, comporte les instructions de structuration suivantes:
- IF ... THEN ... ELSE ... pour l'alternative simple,
- CASE ... OF ... pour l'alternative multiple,
- FOR ... (DOWN)TO ... pour la répétitivité énumérée,
- REPEAT ... UNTIL ... pour la répétitivité jusqu'à la réalisation d'une condition et
- WHILE ... DO ... pour la répétitive tant qu'une condition est vraie.

Pour prendre en compte ces éléments dans la démarche d'analyse, une représentation graphique associée à chacune de ces primitives est définie (cf. figure 2). Il devient ainsi possible de décrire des alternatives et des répétitivités dans l'arbre de décomposition hiérarchique.

Lorsque l'analyse descendante est menée à son terme, il est toujours possible de représenter, directement sous les boîtes terminales, les instructions Pascal associées. Celles-ci peuvent être des opérations de lecture/écriture (READWRITE), d'affectation de valeur à une variable (:=) ou d'appel paramétré d'une fonction ou d'une procédure.

Pour réduire la taille d'un programme, et pour le rendre plus facilement compréhensible, il est important, à chaque fois que cela est possible, d'utiliser des fonctions ou des procédures. L'analyse hiérarchique est bien adaptée pour mettre en évidence le besoin de telles routines. Il apparaît lorsque plusieurs boîtes « action » sont similaires. Le paramétrage de la routine permet de ne préciser qu'au moment de son appel son contexte d'utilisation, et donc de spécifier les caractéristiques propres à l'action appelante.

Afin de les distinguer dans la représentation graphique les actions sont inscrites dans des boîtes aux coins carrés, et les instructions Pascal à l'intérieur de rectangles à coins arrondis. Cette convention est parfaitement logique, avec celle adoptée pour les instructions de structuration. Un arbre complet de décomposition hiérachique d'un problème ressemble à celui de la figure 3.

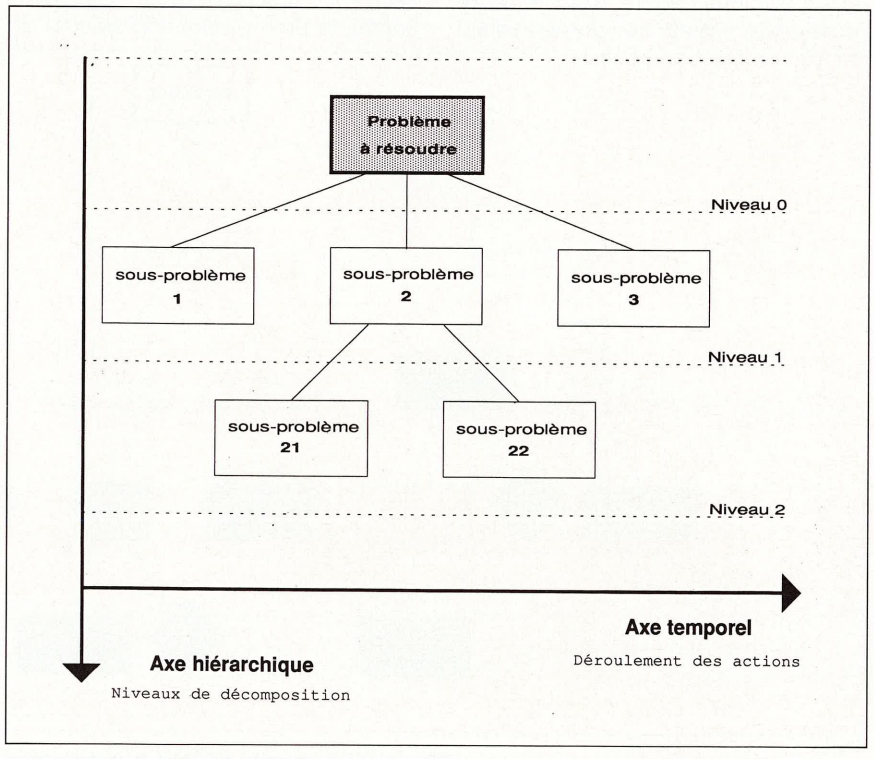


Figure 1 : **Décomposition hiérarchique d'un problème**

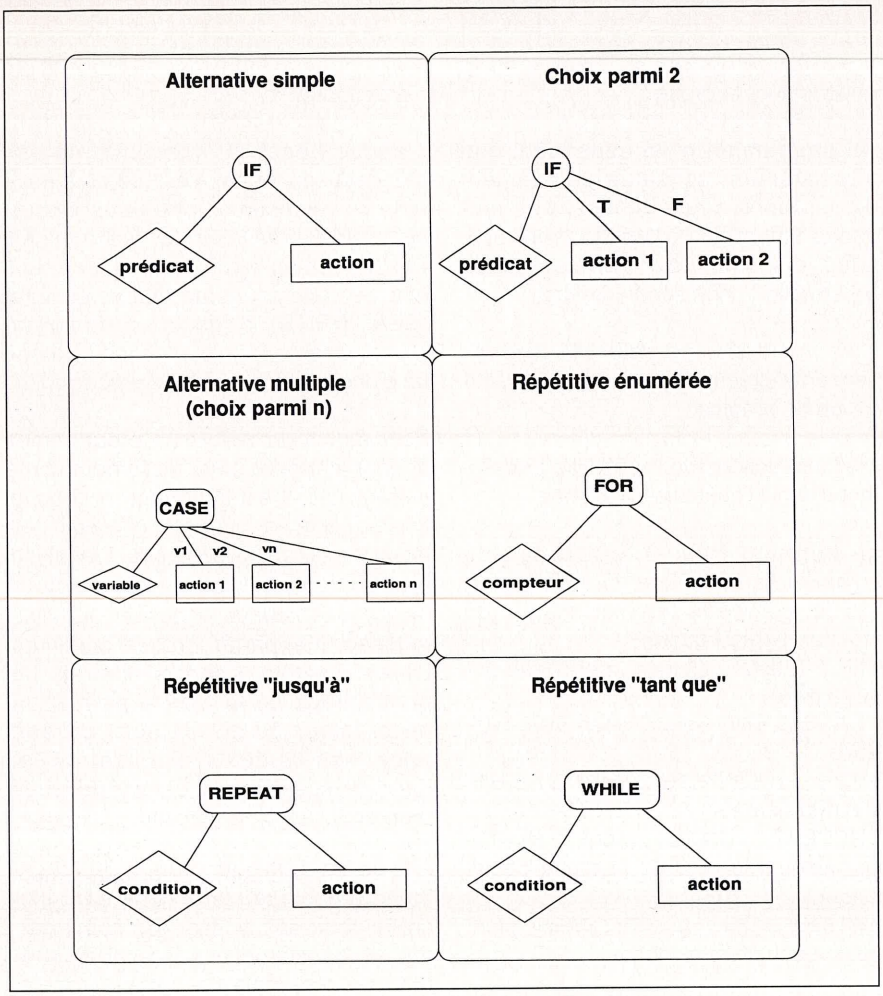


Figure 2 : **Instructions structurées Pascal**

Application à un exemple...

L'application proposée consiste à écrire un programme Pascal qui convertisse un nombre depuis le système de numération romain en son équivalent dans le système arabe, et qui assure également la traduction inverse. L'encadré ci-joint rappelle les règles employées par les romains pour représenter les nombres. Lors de son exécution, le programme doit afficher un menu qui propose à l'utilisateur, soit une conversion « Romain —> Arabe », soit une conversion «Arabe —> Romain », ou bien de sortir de l'application.

L'arbre programmatique du programme principal est présenté sur la figure 4. L'application consiste à itérer l'enregistrement et l'exécution d'une demande de l'utilisateur. Cette itération cesse lorsque celui-ci choisit de quitter au lieu de faire une conversion. La partie « traitement » se compose donc d'une boucle REPEAT ... UNTIL ... qui prend fin lorsque la variable booléenne « Termine » est à la valeur TRUE. Cette variable doit être mise à FALSE dans une partie « Initialisation » exécutée au démarrage de l'application. Lorsque le traitement est terminé, un message d'adieu est affiché.

Le corps de la boucle REPEAT ... UNTIL ... consiste à enregistrer et à traiter une requête de l'utilisateur, c'est-à-dire à exécuter en séquence les opérations:
- afficher le menu des options,
- enregistrer une requête utilisateur,
- traiter la requête,
- attendre une validation utilisateur pour poursuivre.

La première de ces opérations se traduit facilement en Pascal avec des WRITELN. La deuxième s'implémente avec une structure REPEAT ... UNTIL ... qui lit un caractère « choix » au clavier jusqu'à ce qu'il soit dans l'intervalle ['1' .. '3']. Pour traiter la requête

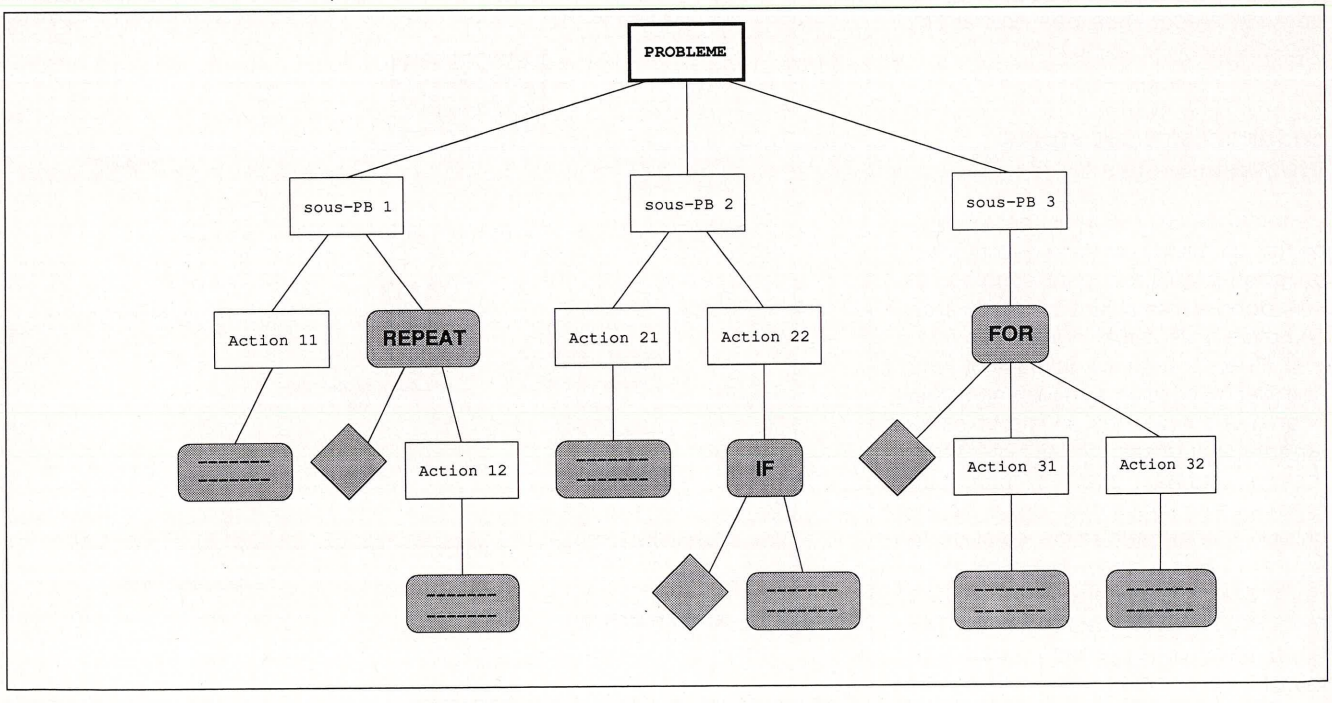


Figure 3 : **Exemple d'arbre programmatique**

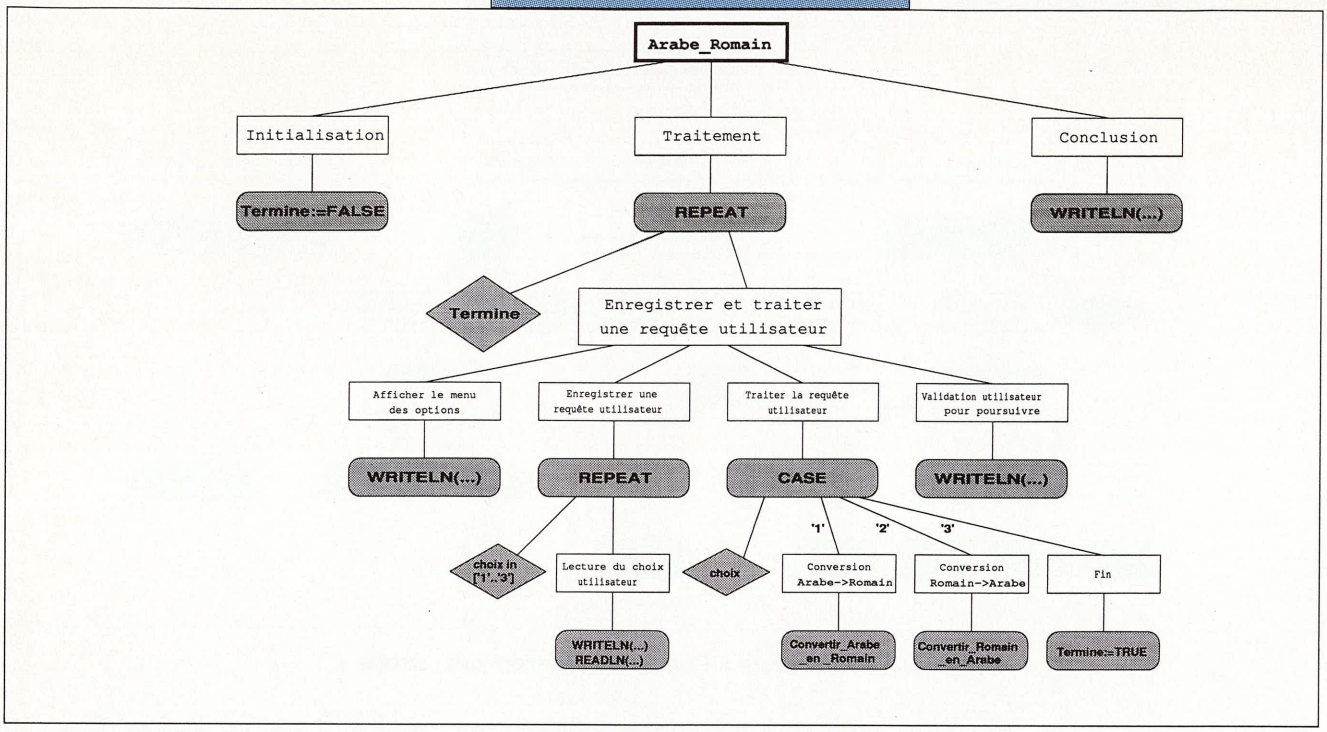


Figure 4 : **Corps du programme de conversion Arabe <- - -> Romain**

sélectionnée, on utilise l'instruction CASE choix OF. Si le souhait de l'utilisateur est de quitter l'application, on positionne « Termine » à TRUE, afin de sortir de la boucle d'itérations. La sélection d'une conversion entraine l'appel d'une des procédures « Convertir_ Arabe_ en_ Romain » ou « Convertir _ Romain_ en_Arabe ». Ce découpage en procédures est fait pour faciliter la compréhension du programme. La dernière opération, d'attente de validation utilisateur pour poursuivre, s'implémente simplement par : WHILE ReadKey=" DO. L'arbre programmatique de la figure 4 traduit de façon synthétique l'organisation complète de l'application.

Les figures 5 et 6 décrivent le fonctionnement des procédures de conversion. Dans les deux cas, le problème initial est décomposé en trois sous-problèmes:

- lecture du nombre à convertir,
- conversion du nombre,
- présentation des résultats à l'écran.

La lecture du nombre se compose d'une boucle REPEAT ... UNTIL ... qui prend fin lorsque la variable booléenne « Nombre_Ok » est à TRUE, c'est-à-dire lorsque le nombre est correctement écrit. Cette itération commence par lire le nombre au clavier, pour ensuite vérifier sa syntaxe. Il est intéressant de remarquer que l'introduction du nombre est presque la même pour les deux procédures de conversion. Dans les deux cas, il s'agit de lire une chaîne de caractères au clavier. La seule différence concerne le domaine

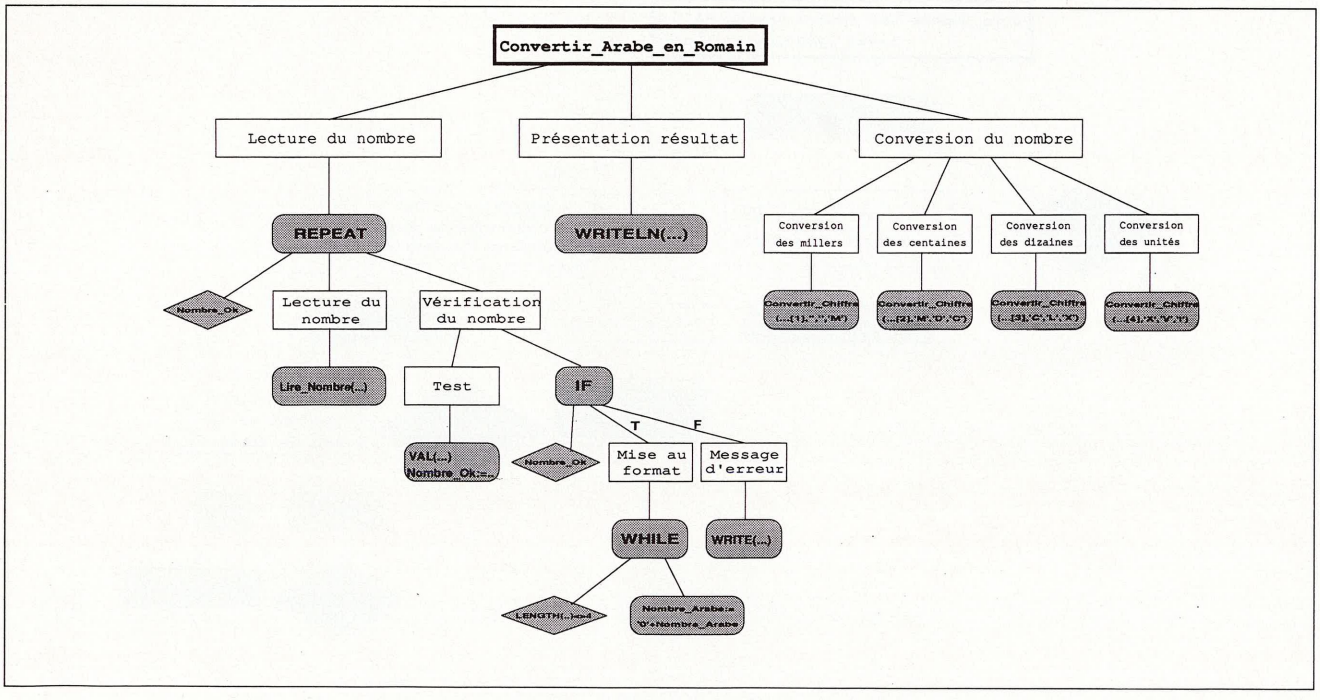


Figure 5 : **Procédure « Convertir_Arabe_en_Romain »**

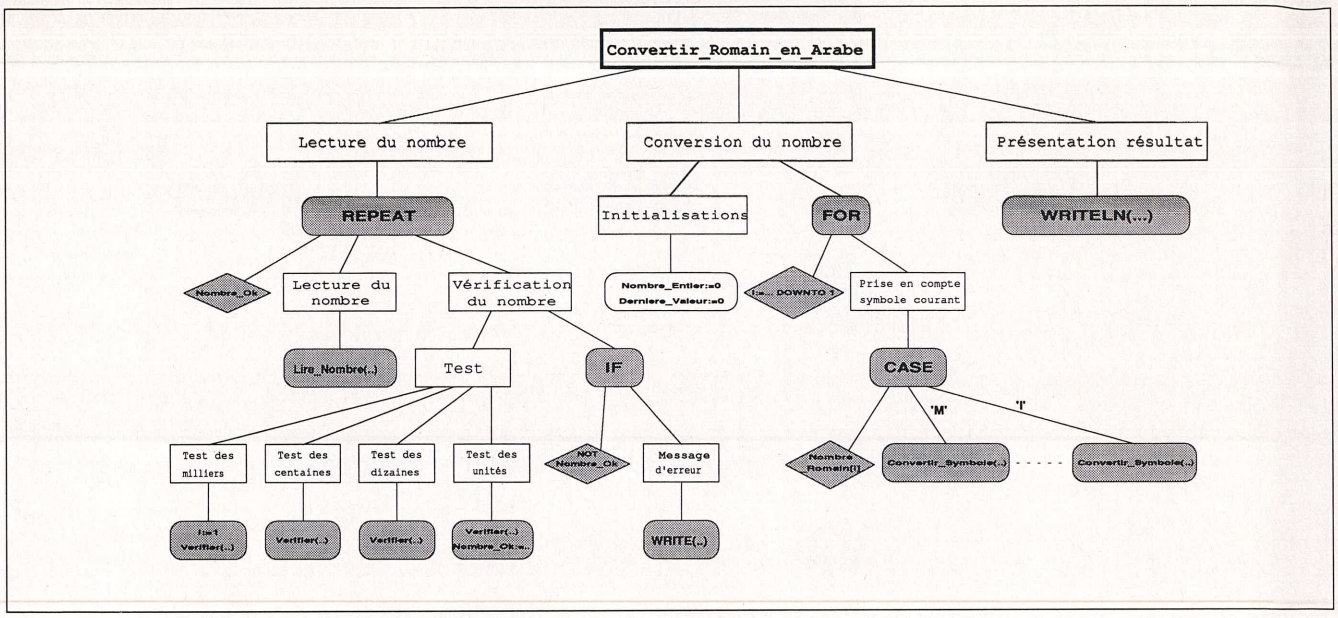


Figure 6 : **Procédure « Convertir _ Romain _ en _ Arabe »**

des caractères autorisés : les chiffres de 0 à 9 pour un nombre arabe, les symboles M, D, C, L, X et I pour un nombre romain. Puisque le langage Pascal permet de définir des types ensembles, il est astucieux de ne définir qu'une seule procédure« Lire_ Nombre ", paramètrée avec le domainedes caractères autorisés. Celle-ci lit une chaîne de caractères au clavier jusqu'à ce que tous les carcatères introduits appartiennent au domaine. Son fonctionnement est résumé sur l'arbre programmatique de la figure 7.

Pour vérifier que le nombre est correctement écrit, la méthode est différente selon qu'il est représenté dans le système arabe ou romain. Dans le premier cas, il est simplement converti sous forme entière avec la procédure Pascal « VAL() », et on vérifie qu'il est compris entre 1 et 3999. Si c'est le cas, le nombre est mis dans le format standard pour la conversion en complétant le début de la chaîne de caractères avec des « 0 » pour toujours avoir quatre caractères. Pour un nombre romain, le test se décompose en test des milliers, des centaines, des dizaines et des unités. En effet, pour chacun de ces digits, la construction est la même, seuls changent les symboles utilisés: I, V et X pour les unités, X, L et C pour les dizaines, C, D et M pour les centaines et M pour les milliers. Ainsi le chiffre 7 positionné en unité se représente VII, et le même chiffre 7 placé en dizaine s'écrit LXX. La figure 8 représente le diagramme syntaxique d'écriture d'un chiffre dans le système romain; ce schéma se lit de la gauche vers la droite. Les différents chemins correspondent aux associations possibles de symboles. C'est exactement ce diagramme qui est implémenté dans la procédure « Verifier() ». Celle-ci fait progresser l'indice « i » dans la chaîne à tester, tant que la syntaxe est correcte. Si au dernier appel de « Verifier », l' indice

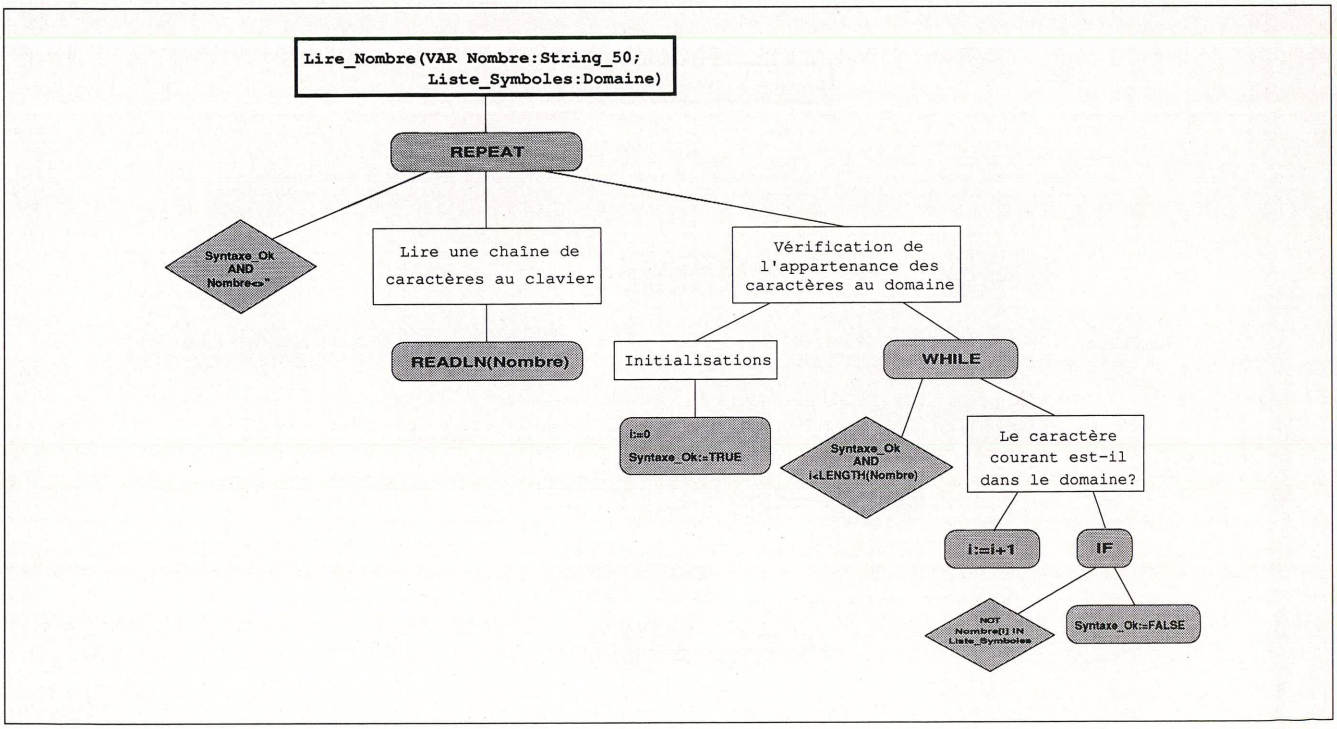


Figure 7 : **Procédure « Lire - Nombre »**

Le système de numération romain

Qui n'a jamais remarqué, gravé sur un monument ou sur une plaque commémorative, un nombre romain? Ils sont souvent utilisés pour exprimer des dates. Notre éducation nous ayant habitués aux chiffres arabes, leur transcription n'est pas toujours facile. Nous allons donc revenir 2000 ans en arrière pour expliquer comment Jules César comptait...

Dans le système de numération romain, les nombres sont générés à partir des bases cinq et dix. Des symboles différents sont utilisés pour représenter chaque puissance de dix. D'autres signes équivalent au regroupement de cinq symboles identiques. Les signes suivants sont disponibles, avec leur valeur associée :
I = 1; V = 5; X=10; L = 50; C = 100; D = 500; M = 1000.

On serait alors en droit de penser qu'un nombre s'exprime par la juxtaposition de tels symboles de telle sorte que la somme des valeurs coincide avec le nombre. Ainsi 1969 aurait « logiquement » pu s'écrire : MDCCCCLXVIIII.
Et bien non! Cela eut été trop simple...

Afin de réduire le nombre de symboles utilisés pour exprimer un nombre (13 dans l'exemple précédent!), les romains ont défini les deux règles suivantes:
- tout signe à droite d'un signe de valeur supérieure s'y ajoute,
- tout signe à gauche d'un signe de valeur supérieure s'en retranche.
Ces règles impliquent qu'il ne peut jamais y avoir plus de trois signes identiques consécutifs dans un nombre. 1969 s'exprime alors correctement sous la forme MCMLXIX.

On remarquera que le nombre zéro est absent de système de numération romain. Ce dernier est par ailleurs très peu commode pour traiter les opérations arithmètiques élémentaires. Il ne saute pas forcément aux yeux que: MCCVXXXIV + DCVXXXI = MCMLXV !!! C'est sans doute pourquoi les mathématiques n'ont que peu progressé sous l'Empire Romain.

Pour ceux que les nombres intéressent, nous pouvons leur conseiller l'excellent ouvrage « Les nombres et leurs mystères » publié par André Warusfel aux éditions du Seuil, dans la collection Points Sciences.

se positionne en bout de chaîne, c'est que le nombre est correctement écrit.

Une fois le nombre lu, il reste à le convertir. Lorsqu'il est représenté dans le système romain, il suffit simplement d'utiliser les règles de construction rappelées dans l'encadré: tout signe à droite d'un signe de valeur supérieure s'y ajoute, tout signe à gauche d'un signe de valeur supérieure s'en retranche. Comme cette règle est identique pour tous les symboles romains, on définit une procédure générique « Convertir_Symbole » paramétrée avec la valeur du dernier symbole et celle du symbole courant. Cette procédure est appelée successivement pour chaque caractère du nombre. A la fin, le résultat est affiché à l'écran par un WRITELN.

Lorsque le nombre est représenté dans le système arabe, on utilise, comme expliqué précèdemment, le fait que la représentation des milliers, centaines, dizaines et unités est identique, à la différence près des symboles utilisés. La procédure « Convertir_Chiffre » transcrit le chiffre demandé avec les symboles passés comme paramètres. Cette procédure est successivement appelée pour le chiffre des milliers, des centaines, des dizaines et des unités. Le nombre romain est alors progressivement affiché à l'écran. Les arbres programmatiques des figures 5 et 6 résument ce qui vient d'être expliqué en plusieurs dizaines de lignes de texte! Elles constituent également un excellent support lors de l'écriture du code; pour

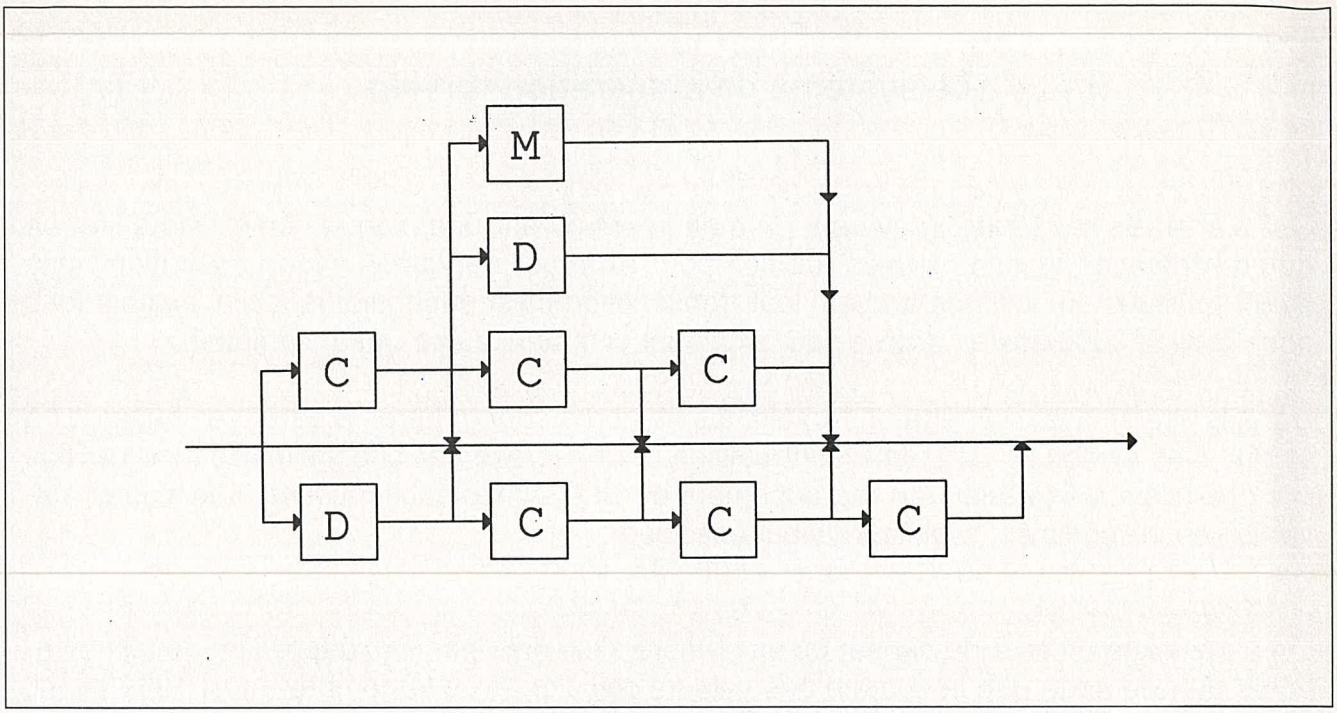


Figure 8 : **Diagramme syntaxique de représentation d'un nombre romain**

preuve, votre serviteur qui les a intensivement employées durant l'écriture du programme ci-joint!

Quelques conseils à retenir...

La construction des premiers arbres programmatiques est toujours difficile: manque d'habitude pour ce mode de réflexion, feuille de papier jamais assez large, nombreuses ratures, etc. Il est toujours préfèrable de réfléchir globalement au problème posé avant de commencer à dessiner le premier arbre. Lorsque l'organisation générale de l'application est décidée, et qu'une première définition des données manipulées existe, la construction des graphiques est beaucoup plus naturelle.

Si les arbres sortent des limites de la feuille de papier, ce n'est aucunement parce que la méthode est mauvaise, mais simplement car l'application est mal structurée!

Il n'est jamais très grave de définir trop de procédures, dans la mesure où leurs rôles sont clairement définis. Il est par contre toujours trés mauvais de ne pas en écrire assez ! Lorsque le code est généré et fonctionne, il est capital de se donner la peine de mettre au propre les arbres programmatiques élaborés durant l'analyse. Si ce travail supplémentaire peut sembler inutile, il se révèlera salutaire lors de la reprise du code après quelques mois d'oubli!

Le lecteur ne pourra alors que se féliciter d'avoir lu ce numéro de DEBUG Magazine. . .

Pour résumer en une phrase l'intérêt des arbres programmatiques, nous affirmons qu'ils constituent un excellent support de réflexion lors de l'écriture du programme, pour se transformer en document de synthèse, une fois le code produit. Que ceux qui ne sont pas encore pleinement convaincus par une telle approche méditent cette citation de Tristan Bernard: « il vaut mieux ne pas réfléchir du tout, que de ne pas réfléchir assez »!

DISQUETTE LISTING
DEBUG N° 12

Algorithmique - Piles, par Frédéric Berton; C
Grilles de mots casés, par Laurent Chambaret; PASCAL
PROTECT & VERIF, par J B Courtois; ASM, C
Mémoire cache, par Alain Deikman (DDJ); C
High Screen 5, programme démo par S Grun; PASCAL
Paradox Engine, programme démo par S Grun; C
Turbo Pascal for Windows, programme démo par S Grun; PASCAL
Gestionnaire de l'imprimante, par Andy Klein (DDJ); C
Arbres programmatiques, par J Ch Riat; PASCAL

NOTION D'ALGORITHMIQUE

"Les structures de données évoluées"

par Frédéric Berton

Dans cet article je vous propose de découvrir certaines structures de données évoluées très couramment utilisées en informatique, nous étudierons le fonctionnement et l'implémentation en langage C des piles.

J'ai volontairement choisi le langage C pour réaliser les différents développements que nous serons amenés à réaliser car ce langage est particulièrement bien adapté à la définition de ce type de données. Toutefois il sera très facile d'effectuer des conversions vers d'autres langages évolués comme Pascal ou ADA. Mais maintenant passons aux choses sérieuses, la première structure de données que nous allons étudier est la pile.

La pile

La notion de pile est très souvent utilisée dans la vie courante, de la pile d'assiettes à la pile de courriers en retard en passant par la pile de dossiers ; les exemples ne manquent pas. Le principe de fonctionnement d'une pile est toujours le même quelque soit sa nature, on empilera des dossiers les uns sur les autres, puis on les dépilera en commençant par le dossier qui se trouve sur le haut de la pile. Ce qui revient à dire que le dernier dossier empilé sera le premier dépilé.

La pile est une structure de données fondamentale en informatique, on l'utilisera pour stocker des paramètres que l'on passe à un sous-programme pour garder la trace des transactions effectuées dans une gestion de transaction, etc...

Dans les piles, l'ajout et la suppression de données se fait toujours par la même extrémité que l'on appelle le sommet (on verra plus loin que d'autres structures de données permettent d'effectuer des opérations équivalentes n'importe où dans la pile, qui ne s'appellera plus une pile mais une liste). Les piles gérées comme nous venons de le voir sont communément appelées pile LIFO pour Last In First Out ou plus simplement Dernière Entrée Première Sortie (DEPS).

Les actions possibles sur une pile sont d'empiler un élément au sommet de celle-ci, dépiler un élément du sommet de la pile (voir note 1), tester si la pile est vide. On peut bien sûr ajouter bien d'autres fonctionnalitées comme dupliquer, comparer, trier, etc... Mais ceci sort du cadre de cet article, et puis il faudra bien que vous occupiez vos longues soirées d'hiver.

Pour implémenter les piles on peut utiliser deux représentations distinctes. La représentation contiguë sous forme de tableau, ou la représentation chaînée mettant en oeuvre les pointeurs pour faire un chaînage des éléments de la pile. Cette dernière méthode bien qu'un peu plus complexe à mettre en oeuvre présente l'avantage d'être dynamique, et donc de ne pas être limitée par un nombre d'éléments figés comme dans un tableau.

Implémentation de la représentation contiguë d'une pile

La représentation d'une pile en utilisant un tableau est certainement la méthode la plus simple que l'on puisse mettre en oeuvre, étant donné la simplicité d'utilisation d'un tableau. Malheureusement elle souffre des limites inhérentes aux tableaux. Effectivement on sera obligé de connaître exactement la taille de la pile lors de sa déclaration dans le programme ; or bien souvent le programmeur n'a qu'une idée très approximative de la taille qu'il faudra pour sa pile. Il sera donc obligé de sur-dimensionner la taille de celle-ci pour ne pas risquer de dépasser la capacité de sa pile (provoquant inexorablement un « core dump de mort »). Il est évident que la simplification du programme se fait au détriment de ses performances surtout au niveau de la taille de l'exécutable.

Comme le montre la figure 1, l'implémentation d'une pile par un tableau indicé, se fait simplement en déclarant un tableau d'une taille déterminée, que l'on appellera Pile. C'est dans ce tableau que seront stockées les données de notre pile. Il faudra parallèlement maintenir un index appelé Sommet qui donnera la position du sommet de la pile.

Lors de la définition de la fonction Empiler() on prendra soin d'ajouter un test destiné à éviter le débordement de la pile. La position d'insertion dans le tableau est obtenue en incrémentant de 1 la valeur de Sommet soit :

```
Pos = Sommet + 1.
```

Pour dépiler un élément, il suffit d'adresser une case du tableau en utilisant comme index la valeur de Sommet soit : . En n'oubliant pas de décrémenter Sommet de 1. Comme vous vous en doutez la fonction Dépiler() devra contenir un test évitant de dépiler plus bas que la limite inférieure de la pile (sinon vous tombez).

```
Élément = Pile[Sommet[
```

Vous pourrez constater que dans une représentation contiguë les éléments dépilés ne sont perdus que logiquement, physiquement ils sont toujours stockés dans la pile au même endroit. Ils ne seront réellement perdus que si vous empilez de nouvelles informations, écrasant ainsi les anciennes.

A l'issue de ces quelques explications il est évident que tout le travail de gestion de la pile est réalisé grâce à la bonne utilisation de la variable Sommet.

Les fichiers PILECONT.C et PILECONT.H vous présentent un exemple d'implémentation de la représentation contiguë d'une pile. Le fichier PILECONT.H contient la déclaration du type de données pile contiguë (PileCont_t) qui est une structure appelée PileCont_s. Cette structure correspond à la figure 1 ; elle contient un entier non signé (Sommet) qui donne l'adresse du sommet de la pile (Pile) qui est représentée par un tableau indexé. Le nombre d'éléments de Pile est spécifié par la constante MAXPILE. Attention l'élément 0 du tableau n'est pas utilisé. Lorsque la variable Sommet contient 0 la pile est vide. Ce fichier contient également les déclarations des différents codes erreurs que peuvent retourner les fonctions définies, et les déclarations des prototypes de fonctions.

Implémentation de la représentation chaînée d'une pile

Voyons maintenant comment implémenter une pile grâce à la représentation chaînée. Beaucoup de détracteurs (voir note 2) du langage C le critiquent car ils prétendent que l'utilisation des pointeurs rend ce langage trop dur à lire et difficile à maintenir. J'espère que les lignes qui vont suivre vous persuaderons du contraire. La figure 2 reprend le même schéma que la figure 1 sauf que les différents éléments de la pile sont liés les uns aux autres par des pointeurs (symbolisés par des cases dont partent des flèches).

Le principe de fonctionnement est le même que celui que nous venons de voir pour la pile contiguë, à quelques différences prés. La structure PileChaîne_t contient le nombre d'éléments total de la pile dans une variable de type entier non signé NbreElem (si cette valeur est 0 alors la pile est vide), et un pointeur de type Noeud_t * pSommet (si ce pointeur est égal à NULL alors la pile n'existe pas, ou est vide pour l'instant) sur l'élément de la pile qui est au sommet de celle-ci. Les différents éléments de la pile quant à eux sont de type Noeud_t. Un noeud contient une donnée Donnée (dans notre exemple un entier long) et un pointeur sur le noeud suivant de type Noeud_t * pNext. Chaque élément de la pile pointe sur l'élément suivant assurant ainsi Le chaînage des éléments entre eux. Pour manipuler la pile on utilisera toujours une variable de type PileChaîne_t qui contient le poin-

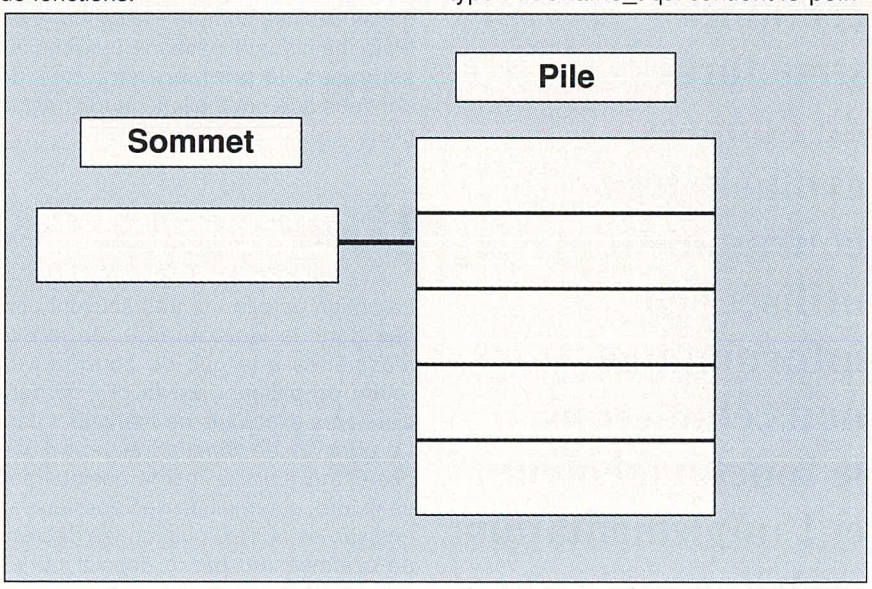


Figure 1 : **Représentation contiguë d'une pile.**

teur pSommet sur le sommet de la pile. On n'utilise donc jamais directement les éléments de la pile on passe toujours par le pointeur sur le sommet comme dans la Représentation contiguë. La fonction Empiler() définie dans PILECHAINE.C assure trois fonctions principales : premièrement elle alloue de la mémoire aux éléments à empiler en créant un pointeur de type Noeud_t*, deuxièmement elle assure le chaînage des éléments entre eux, et troisièmement elle effectue la gestion de l'entête de pile.

La fonction Dépiler() récupère la donnée qui se trouve au sommet de la pile en utilisant le pointeur sur le sommet (pSommet) spécifié dans l'entête. Le nombre d'éléments de la pile (NbreElem) est décrémenté, puis le pointeur sur le sommet est mis à jour pour pointer sur l'élément de la pile précédant le sommet. En dernier lieu la mémoire allouée pour l'élément récupéré est libérée. Contrairement à la représentation contiguë ou les éléments dépilés ne sont perdus que logiquement. Dans la représentation chaînée ces éléments sont perdus physiquement puisque la place mémoire qui leur est attribué par la fonction Empiler(), est restituée au système par la fonction Dépiler().

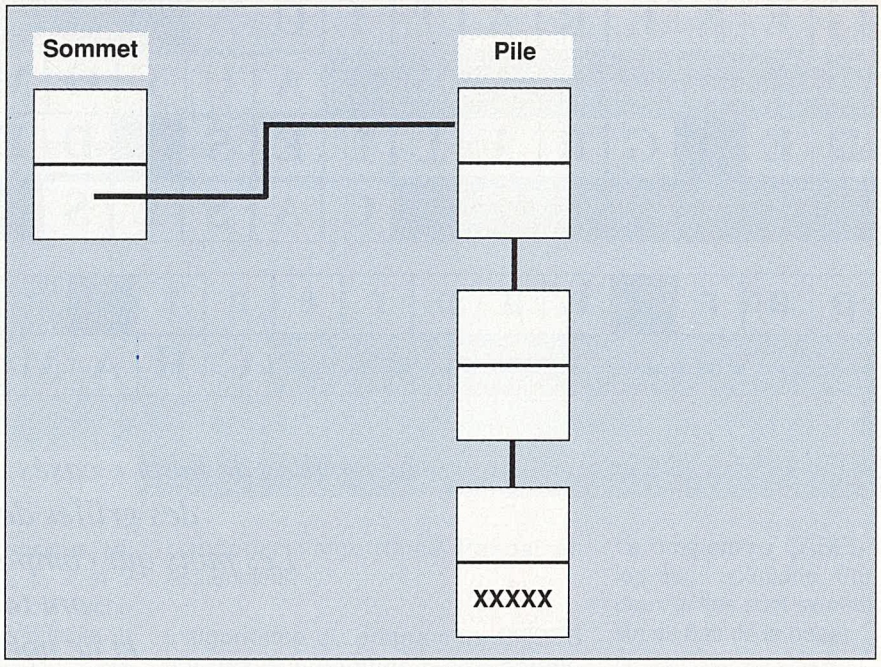


Figure 2 : **Représentation chaînée d'une pile.**

Conclusion sur les piles

Nous venons de voir deux possibilités d'implémentation d'une pile. Comme vous l'avez sûrement remarqué la donnée que nous avons stocké dans notre pile était toujours un entier long, vous devez penser que pour une structure évoluée nous sommes bien limités. En fait j'ai volontairement limité mes exemples à ce type de données pour simplifier mes démonstrations. Dans une utilisation réelle la structure Noeud_s aurait plutôt la forme suivante:

```
struct Noeud_s
{
 void           *pData;   /* Pointeur sur une donnée non typée */
 unsigned int    Size;    /* Taille de la donnée pointée par pData */
 struct Noeud_s *pNext;  /* Pointeur sur le noeud suivant */
};
```

Il suffirait de faire quelques modifications dans les fonctions Empiler() et Depiler() pour gérer ce type de structure. Nous aurons l'occasion d'en reparler lorsque nous étudierons les listes chaînées dans notre prochain article.

Je serai heureux de recevoir toutes vos remarques ainsi que les différents travaux que vous aurez réalisés, que le journal se chargera de me transmettre. Quant à moi je m'engage à vous répondre, quoi de plus naturel. En attendant bon travail...

Frédéric Berton est Analyste-Programmeur. Il programme en Langage C sous Windows.

Notes

1) Il est important de noter que l'action de dépiler une information entraîne la perte de celle-ci au niveau de la pile, sinon on ne peut pas dépiler puisque l'élément reste toujours sur la pile.

2) Certainement des gens qui ont trop abusés du COBOL, qui comme tout le monde le sait, cause des dégâts irrémédiables et irréversibles au cerveau en cas d'utilisation prolongée. D'ailleurs ce langage n'est pas remboursé par la sécurité sociale!

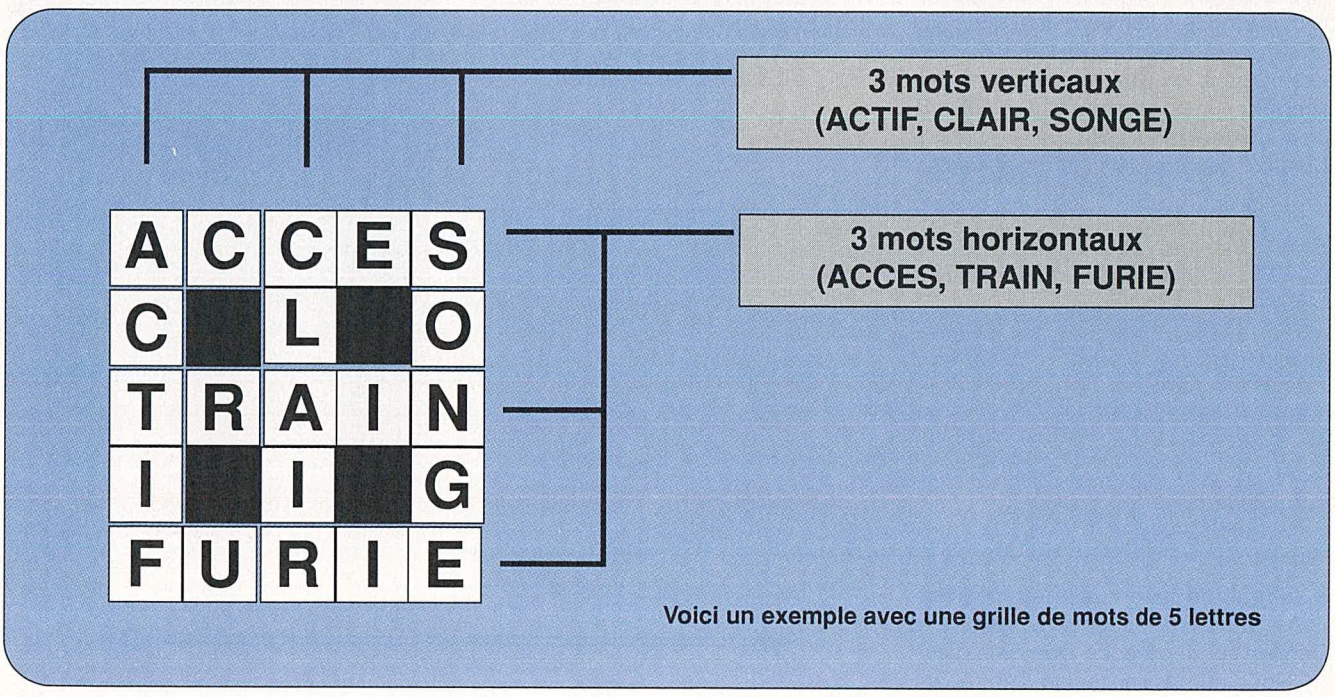
GÉNÉRATION AUTOMATIQUE DE GRILLES DE MOTS CASÉS

par Laurent CHAMBARET

Les grilles de mots « casés » sont une variante structurée des grilles de mots croisés traditionnelles. Les mots qui composent une grille de mots casés ont tous le même nombre de lettres, et ce nombre est impair (3, 5, 7, 9 etc). La grille est carrée, et les mots sont placés régulièrement, une ligne sur deux, et une colonne sur deux.

L'objectif que l'on se fixe est de réaliser un programme qui, à partir d'une liste de mots, génère automatiquement toutes les grilles de mots casés possibles. Posé ainsi, ce problème est typiquement du ressort de l'intelligence artificielle, avec une base de connaissance (les mots du dictionnaire) et un moteur d'inférence (l 'algorithme qui va valider les bonnes grilles). Cependant, le problème est suffisamment restreint pour être résolu avec un langage de programmation structurée traditionnel.

Le programme explore l'intégralité des combinaisons de mots. Bien sûr, il arrête les recherches quand il constate une impossibilité lors de la mise en relation des mots. Plus les mots sont longs (7 lettres, 9 lettres ...) et plus il y a de mots à mettre en relation pour former une grille. Le nombrede mots disponibles dans la base de données entre aussi en ligne de compte. Si il y a beaucoup de mots, alors il y a plus de recherches à effectuer, mais il y a aussi plus de chances de trouver une correspondance de lettres. Il faut d'ailleurs remarquer que les correspondances se font toujours sur les lettres de rang impair : la 1ère, la 3ème, la 5ème, etc.



[SERVICE LECTEURS : INTERESSANT = 27 / MOYEN = 28 / MAUVAIS = 29]

Lettres par mot		Correspondance lettres par mot	Mots par grille	Mots dans un dictionnaire
3		2	4	44
5		3	6	354
7		4	8	654
9		5	10	675

Avec un éditeur de texte classique, j'ai saisi tous les mots de 3, 5, 7 et 9 lettres dans un dictionnaire de poche de 500 pages. C'est moins difficile et moins long qu'il n'y paraît. Cela demande seulement beaucoup de patience et d'attention. Il faut saisir les mots dans des fichiers séparés (3.MOT, 5.MOT, 7.MOT, 9.MOT) avec un mot par ligne, en minuscule ou en majuscule. Les accents ne doivent pas être pris en compte. En fait, ce sont les grilles de 7 lettres qui s'avèrent les plus intéressantes. L'avantage d'un dictionnaire de poche est de présenter uniquement des mots courants du langage français, ainsi on obtient des grilles de mots croisés cohérentes. L'idéal serait d'avoir accès à une base de donnée dictionnaire des mots français, mais celles-ci sont toujours codifiées par leurs concepteurs.

Voici par exemple le début du fichier 7.MOT :

abattre
abjurer
aborder
abréger
abroger
abrutir
accoter
accuser
. . .

La puissance pure de votre micro-ordinateur, c'est à dire sa vitesse de traitement en mémoire centrale, sera mise à forte contribution. Un PC-AT à 12 MHz est suffisant pour trouver toutes les grilles de mots de 3, 5 et 7 lettres, mais pas les grilles de 9 lettres. Pour celles-là il faudra faire appel à un 386, ou faire tourner l'ordinateur plusieurs heures, en espérant qu'il existe bien des solutions de grilles de 9 lettres!.La meilleure solution consiste plutôt à transcrire l'algorithme Pascal en Assembleur.

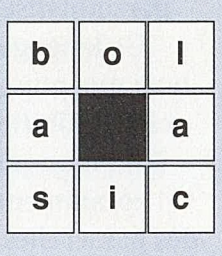
L'algorithme du programme CASES n'a pas la prétention d'être parfait, mais il correspond au raisonnement logique que l'on applique soi-même quand on cherche à réaliser de telles grilles de mots croisés : on place arbitrairement un mot en haut de la grille, puis on cherche un mot qui commence par la même lettre et on le place verticalement à gauche de la grille. On cherche alors un mot qui commence par la troisième lettre de ce mot et on le met horizontalement à cette position. Ça se complique ensuite car il faut chercher un mot qui commence par la 3ème lettre du 1er mot horizontal et qui possède une 3ème lettre identique à la 3ème lettre du 2ème mot horizontal. Comme on le voit, le système est appliqué alternativement à un mot horizontal puis à un mot vertical, c'est pourquoi le programme possède des procédures ExplorerX et ExplorerY similaires. Chacune de ces procédures fait appel à l'autre. Cette récursivité indirecte s'arrête quand la grille est complète ou quand il n'y a pas de solution. La variable structurée Lettre est un index qui permet d'accéder rapidement aux mots commençant par une lettre déterminée. Il faut noter qu'une grille de mots croisés peut être symétrisée.

Le programme CASES valide toujours les deux solutions. Par exemple, ces deux grilles sont symétriques (voir dessin au bas de la page).

Le programme pose 3 questions :

1) Nombre de lettres par mot ?. Le fichier des mots doit posséder le même nom et l'extension MOT (3.MOT, etc).
2) Faut-il afficher les recherches ? Cet affichage permet de bien comprendre le principe de l'algorithme de recherche, mais il ralentit l'exécution.
3) Faut-il enregistrer les solutions ? Les grilles complètes seront alors sauvegardées dans un fichier d'extension GRI (3.GRI, etc).

Le programme est écrit pour la version 5.5 de Turbo-Pascal, mais il suffit de modifier les options de compilation définies au début du programme pour l'adapter aux versions 4 et 5. Pour l'anecdote, sachez que le problème de la programmation des grilles de mots casés m'avait été proposé par Guy Hachette lui- même, célèbre créateur des mots croisés, fort sympathique d'ailleurs. Les grilles de mots « casés » ne sont qu'un point d'entrée vers la recherche d'algorithmes pour la résolution de problèmes plus complexes, et notamment les grilles de mots croisés non structurées ...



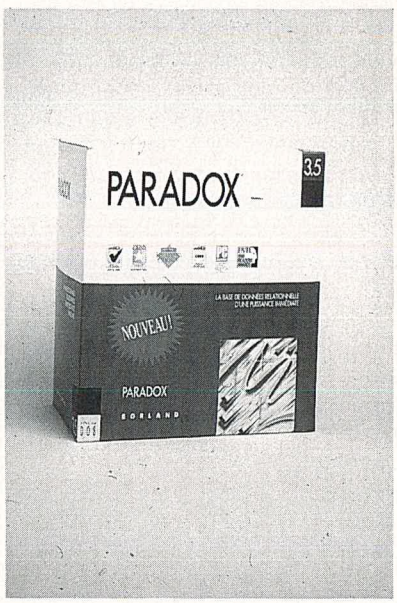
Paradox Engine 1.0

Le lien entre Paradox et C

par Serge Grun

Paradox, de Borland, est un Système de Gestion de Bases de Données (SGBD) relationnel et multi-utilisateurs. Il se démarque de ses concurrents par l'existence de versions SQL, OS/2 et 386 (en plus de la version de base), par l'absence notoire de bogues (vous voyez à qui je pense...) et par une simplicité d'emploi extrême. Paradox dispose d'un langage de programmation interprété appelé PAL, permettant la création d'applications personnalisées. Ce langage, bien que puissant, a ses limitations; il lui manquait, notamment, des passerelles vers des routines externes.

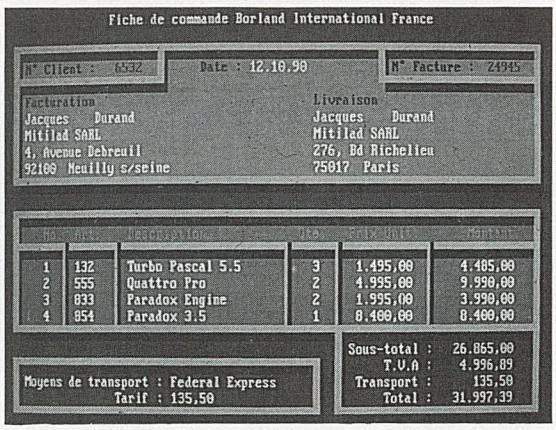
Les limitations du PAL sont aujourd'hui levées grâce à un nouvel outil de développement édité par Borland: le Paradox Engine 1.0 est une librairie C (compatible TurboC et Microsoft C5.1) qui offre, sous forme de 94 fonctions, tout ce qui est nécessaire pour accéder aux tables Paradox à partir d'un programme C (une version Turbo Pascal serait en cours de développement). Ces fonctions permettent de créer, de détruire ou d'indexer des tables, de lire, d'écrire et de rechercher des enregistrements (en assurant, au besoin, la conversion des données) ou encore de gérer les mots de passe et le cryptage des données. Paradox Engine assure également l'intégration d'une application dans un environnement réseau (7 types de réseaux sont reconnus) en permettant le verrouillage implicite ou explicite d'enregistrement ou de tables entières lors des consultations ou des mises à jour.



LE NOYAU COMPACT

Le noyau du Paradox Engine est remarquablement compact: comptez quelques 70 Koctets (avec les messages d'erreur) auxquels il faut ajouter les fonctions employées. L'implémentation de la librairie est d'une cohérence à toute épreuve. Qu'il s'agisse de la

dénomination des fonctions ou des méthodes de passage des paramètres aux fonctions, tout est basé sur le même modèle. Anodin en apparence, ce fait prend toute sa signification lorsqu'il s'agit de comprendre ce qui se passe dans un listing de plusieurs dizaines de pages. J'ai cependant été étonné par la conception des fonctions de recherche: elles font la différence entre majuscules et minuscules, sans qu'il soit possible de modifier ce comportement. Une telle conception est, pour le moins, inhabituelle et regrettable, forçant parfois le programmeur à recourir à des manipulations inutiles. C'est, à mon avis, la plus grosse lacune de Paradox Engine.



Borland Paradox Engine

MANUEL

Un survol du manuel (250 pages) permet de se faire une idée rapide des possibilités de Paradox Engine, mais la partie consacrée à la description des fonctions s'avère être dans certains cas trop superficielle et ce d'autant plus que ce manuel a dans son ensemble un air de je-ne-sais-quoi d'inachevé. Ce n'est sans doute pas une traduction intégrale, mais le manuel a pourtant cette concision des ouvrages anglo-saxons dont la langue de Molière s'accommode si mal. Pour citer un exemple, dans la description des fonctions de recher-che, j'ai trouvé la phrase suivante :

«... Si vous utilisez le modificateur CLOSESTRECORD, l'enregistrement en cours deviendra l'enregistrement avec la valeur supérieure la plus proche de celle recherchée ». Malgré tous mes efforts, je n'ai trouvé nulle part ailleurs d'autre description de ce modificateur. En fait, le manuel ne dit même pas quel paramètre des fonctions de recherche accepte ce modificateur et il a fallu que je disséque le fichier pxengine.h pour trouver la réponse!

UN PROGRAMME EN DEUX HEURES

En dépit d'une documentation qui n'est pas tout à fait à la hauteur de celles auxquelles Borland nous avait habitués, l'utilisation pratique du Paradox Engine est simple, d'autant plus que le code source, abondamment commenté, de deux applications est fourni en exemple. Le petit programme illustré par le listing1 affiche d'abord la structure d'une table Paradox puis, un à un, les enregistrements qu'elle contient. Il a été écrit en TurboC au bout de 2 heures. C'est dire combien l'utilisation des fonctions du Paradox Engine est aisée.

Cependant, ne vous y trompez pas, ce n'est pas un outil à mettre entre toutes les mains. Contrairement à, par exemple Clipper (pour dBase), Paradox Engine n'est pas un produit visant à remplacer Paradox et PAL: il est plutôt conçu comme un complément. Paradox Engine et PAL ont tous les deux leurs limitations, mais les avantages de l'un pallient les inconvénients de l'autre. Par exemple, PAL ne sait pas accéder aux périphériques (terminal de caisse enregistreuse, crayon optique, lecteur de carte de crédit, etc.), ce que Paradox Engine peut parfaitement faire grâce au C. En revanche, Paradox Engine a beaucoup de mal à extraire des données en fonction d'un critère autre que l'identité exacte: manifestement, une telle extraction doit être faite par PAL.

Ces contraintes font que l'utilisation de Paradox Engine seul, pour la réalisation d'une application complète sera, sinon irréaliste, du moins très difficile. En revanche, les développeurs PAL ont maintenant un outil qui leur permet de s'attaquer à virtuellement n'importe quelle réalisation. Le produit pourrait être à mon sens amélioré par l'ajout d'une interface vers le TurboPascal, par l'extension des fonctions de recherche et par une refonte du manuel mais, dans son état actuel, c'est déjà un outil comme on aimerait en voir plus souvent -gageons que le développement PAL va prendre un sacré coup de fouet.

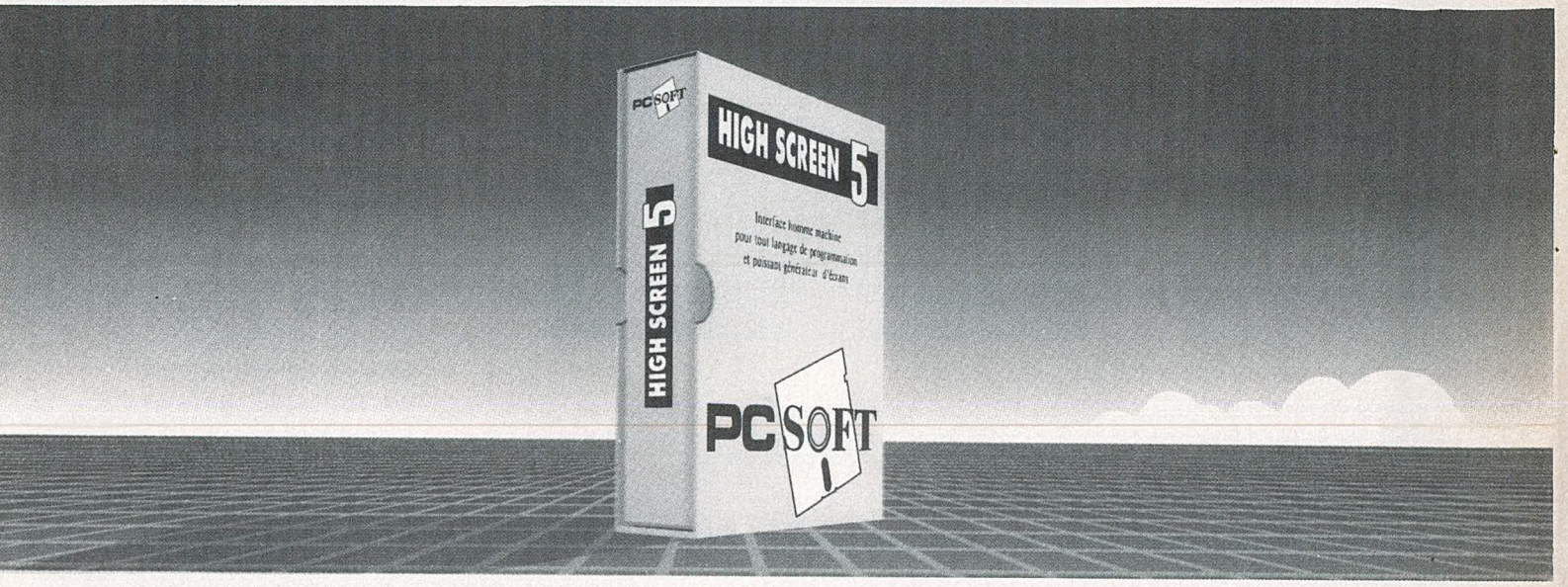
BORLAND
43, av de l'Europe - BP 6
78141 Velizy Cédex
Tél : 39 46 96 69
Prix : 1 995 F. HT.

POINT DE VUE DE BORLAND

La stratégie de Borland est de proposer une totale interopérabilité entre ses applications bureautiques (Paradox, QuattroPro) et ses applications des Turbo langages orientés-objets (TurboC++ et TurboPascal).

Paradox Engine est un produit dont l'objectif est d'être complémentaire au langage de développement PAL du SGBD Paradox. Il représente une passerelle vers des programmes C voulant accéder aux tables Paradox.

Les remarques que vous faites, sont pour la plupart prises en compte pour la future version de Paradox Engine. Celle-ci permettra d'ouvrir encore plus l'architecture de Paradox aux programmes développés dans d'autres langages que le C et dans de nouveaux environnements de travail.



High Screen 5

La dernière mouture à la hauteur de sa réputation.

par Serge Grun

Les générateurs d'écrans n'ont pas une très bonne réputation auprès des programmeurs. La raison en tient peut-être à ce que, outre le manque de contrôle sur le produit final qui leur est souvent reproché, le plus souvent, il faut disposer d'un générateur différent par langage et parfois, pour unmême langage, d'un générateur différent par compilateur. HighScreen5 (HS5), conçu par PCSOFT (une sociétéfrançaise! Cocorico...) apporte enfin une solution élégante à ce problème.

[SERVICE LECTEURS : INTERESSANT = 33 / MOYEN = 34 / MAUVAIS = 35]

HS5 est un produit impressionnant : 3 manuels, 6 disquettes de 1.2Moctets (ou 9 disquettes de 744Koctets), près de 3 Moctets de code pour les générateurs et 300 Koctets par langage installé! Heureusement, toute l'installation est prise en charge par un programme qui permet de spécifier les modules que l'on veut copier.

Cette quantité de code tient à ce que HS5 est bien plus qu'un simple générateur d'écrans. Le produit complet réunit un éditeur d'écrans, un générateur de menus déroulants, un gestionnaire de fenêtres, un gestionnaire de saisie, un outil de prototypage et encore quelques autres accessoires. Le tout fonctionne quel que soit le mode vidéo en cours (texte ou graphique) et surtout, quel que soit le langage utilisé.

L'éditeur d'écrans est le pivot central de tout le système et bien que chaque outil puisse être invoqué séparément, tous sont appelables à partir de l'éditeur. Tous les modules ont une interface utilisateurs articulée autour de menus déroulants et qui tire parfaitement parti de l'utilisation de la souris. L'organisation des menus varie peu d'un module à l'autre, ce qui permet de maîtriser l'ensemble du produit en peu de temps.
Le générateur d'écrans est l'outil central de HS5. Il permet non seulement de dessiner de façon interactive les écrans d'une application, mais aussi d'y placer les champs de saisie et leurs conditions de validation et de créer des fenêtres. Le moteur « run-time » de HS5 est en effet en mesure de gérer la saisie de données, en utilisant des champs, des masques et des contraintes de valeurs prédéfinies lors de la création ou ajustées en cours d'exécution de l'application finale. Toutes les opérations de saisie peuvent se faire en mode plein écran, et la gestion de la souris est entièrement prise en charge par le run-time.

Le générateur d'écrans crée, d'une part, un fichier de description d'écran, permettant de reprendre plus tard le même écran et, d'autre part, un fichier binaire destiné à être utilisé par l'application finale. L'utilisation d'un tel fichier est extrêmement simple:

```
WriteLn(Chr(19)+Chr(254)+Chr(1),'UTILISE,DEMO.AID',Chr(1));
```

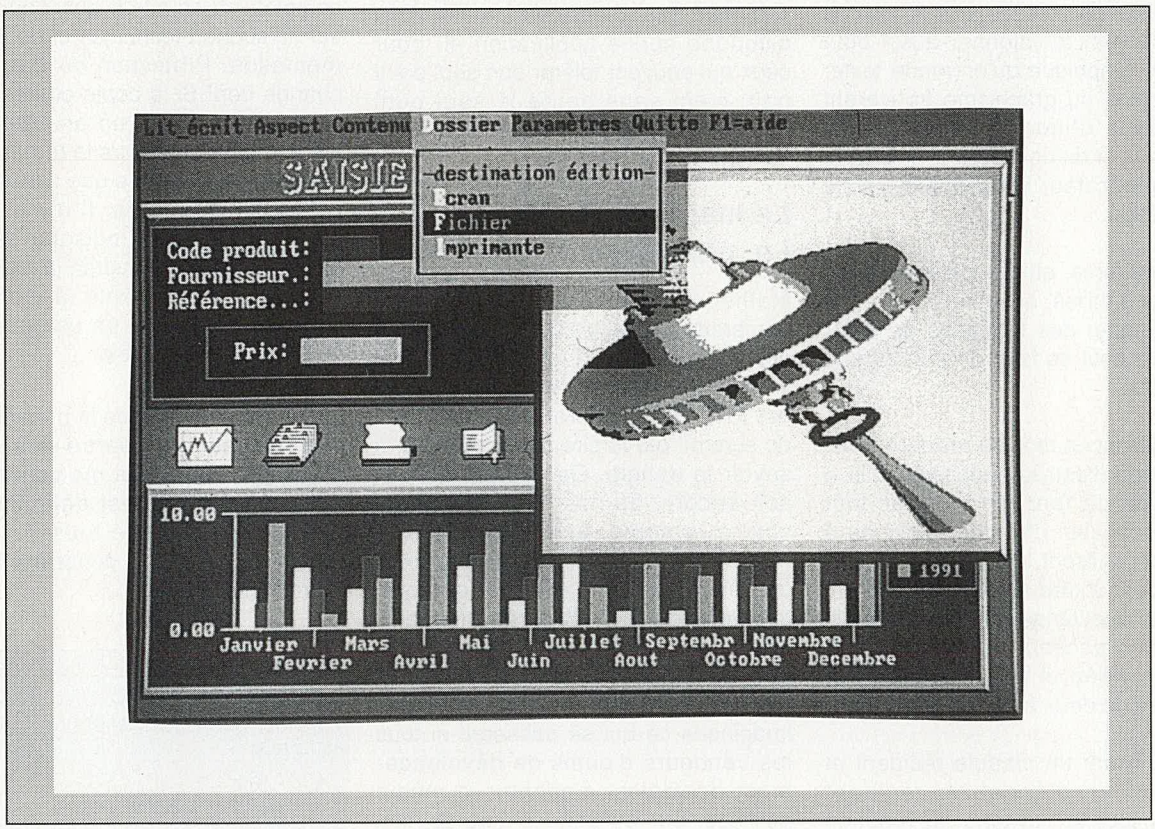
Cette instruction Pascal ouvre le fichier DEMO.AID, affiche l'écran correspondant et prépare la saisie - le tout en une seule ligne ! Le listing qui accompagne cet article démontre la facilité avec laquelle il est possible de réaliser l'affichage et la saisie d'un ensemble de champs. Mais, avec un peu dechance, vous n'aurez même pas ça à écrire, puisque le générateur d'écrans est capable de créer le code source à votre place (c'est d'ailleurs lui qui a crée ce listing). Il faut pourtant mentionner au passage que ce générateur semble avoir quelques problèmes avec la syntaxe de certaines de ses propres fonctions: lors de la création d'un petit exemple en Turbo Pascal, les appels des fonctions de conversion de chaînes de caractères en entiers ne correspondaient pas à la syntaxe déclarée.

HS5 reconnaît six types de champs (texte, numérique, date, liste, booléen, sélecteur et bouton), compatibles avec la définition SAA/CUA d'IBM. En mode graphique, les éventuelles icones d'un écran (s'il y en a) peuvent être traitées comme des boutons. Toutes les interactions entre l'utilisateur et ces champs sont gérées par HS5.

La saisie peut se faire champ par champ ou, au contraire, par écrans entiers. Dans tous les cas, l'application finale peut demander une interruption avant où après la saisie d'un champ ou modifier en cours d'exécution les attributs d'un champ.

Éditeur de menus

L'éditeur de menus fonctionne d'une manière originale: au lieu de dessiner un par un les menus déroulants successifs, les commandes sont saisies



les unes à la suite des autres, dans un éditeur plein écran. C'est l'indentation relative des commandes qui délimite le menu principal et les sous-menus successifs: c'est à la fois simple et efficace.

HS5 gère aussi bien les menus déroulants à la Borland que les menus horizontaux (type Lotus) ou superposés dans des fenêtres rectangulaires. Il reconnait les zones variables dans les libellés et peut associer à chaque commande une ligne de commentaire qu'il affichera sur la dernière ligne de l'écran lors de l'exécution de l'application. Le run-time sait aussi gérer les touches associées aux commandes. Il autorise notamment le choix d'un caractère du libellé et l'appel de la commande associée par la touche alphanumérique correspondante.

L'outil de prototypage

Un outil de prototypage d'applications est inclus avec HS5. Il permet de définir une maquette de l'application finale, de créer tous les écrans nécessaires, de définir leur enchaînement et de générer en fin de compte le code source correspondant. Le programme ainsi créé contiendra tout ce qui est nécessaire pour gérer la partie interface utilisateur et il ne reste plus qu'à y ajouter la partie traitement de données.

Les applications créées à l'aide de ces outils peuvent fonctionner aussi bien en mode graphique qu'en mode texte. Les adeptes du graphisme trouveront un éditeur d'icones (et non pas « icônes ») et de pointeurs souris, ainsi qu'un générateur de graphiques de présentation.

Le tour de force, qu'est l'indépendance du langage utilisé, est obtenu grâce au mode d'appel des fonctions de HS5. Cet appel peut se faire de trois façons différentes:

a. En utilisant un module résident: c'est la méthode illustrée ci-dessus. Elle a l'avantage de fonctionner avec tout langage capable d'afficher du texte à l'écran en utilisant les interruptions du BIOS (c'est à dire, avec virtuellement n'importe quel langage). Par contre, si votre code accède directement à la mémoire vidéo, il vous faudra opter pour l'une de deux autres techniques.

b. En utilisant un module résident et une fonction d'appel: le résident contient tout le run-time; la fonction d'appel ne fait que générer une interruption soft qui est interceptée par le run-time.

c. En incorporant tout le run-time dans l'application: tous les appels se font à travers une fonction interface.

Les deux dernières méthodes supposent que le langage utilisé accepte des modules OBJ extérieurs. PCSoft fournit par ailleurs un certain nombre de modules extérieurs pour les compilateurs qui ne génèrent pas de fichiers OBJ (comme, par exemple, Turbo Pascal).

La première solution est la plus économique en termes de mémoire: le module résident occupe env. 50K et peut être partagé par plusieurs applications. Il oblige, en revanche, à distribuer plusieurs fichiers et à jongler avec les résidents.

La dernière solution est très gourmande en mémoire: 110K pour un test qui affiche un écran et effectue la saisie et la validation de 5 champs ! En revanche, elle permet de réduire le nombre de fichiers à distribuer, puisque un utilitaire permet de réunir les différents écrans et menus d'une application en une librairie unique.

HS5 est donc un produit puissant et relativement facile à maîtriser. L'étendue de ses possibilités le rend suffisamment polyvalent pour s'attaquer à n'importe quelle application et, pour ceux qui peuvent tolérer son seul point noir, c'est sans doute le seul outil nécessaire pour développer une interface utilisateur complète.

Le mal (in)évitable : Le plombage

Malheureusement, en dépit de ses nombreuses qualités, HS5 souffre d'un défaut majeur qui, à mes yeux, le rend inutilisable pour tout travail sérieux: il est PROTEGE contre la copie. Il l'est de surcroît par la pire des méthodes, à savoir le dongle. Qu'une application soit encore, de nos jours, protégée contre la copie, cela, je peux à la rigueur, le concevoir. Mais qu'un outil de développement, écrit par des programmeurs pour d'autres programmeurs, le soit, est une situation inadmissible, quelle que soit la qualité du logiciel. Arrêtons-nous un instant et imaginons ce qui se passerait si tous les vendeurs d'outils de développement décidaient d'adopter la même politique. Sur les disques durs de mon AT386, il y a TurboC++, ZortechC++, TurboPascal5.5 et 6.0, MODULA 2, le MicrosoftC6.1, plusieurs librairies C et Pascal, le kit de développement Windows 3.0, Paradox Engine1.0, Clipper et encore quelques autres outils et compilateurs dont le nom m'échappe. Ces mêmes outils de développement sont installés sur mon vieil AT286. En supposant que chacun de ces produits soit « plombé » de la même façon (et en admettant que la longueur moyenne d'un dongle soit de 3.5 cm), il y aurait, derrière ma machine, un assemblage d'environ 35 centimètres de long, couronné par un câble d'imprimante et amorçant, sous l'effet de la pesanteur, une disgracieuse parabole en direction du sol. Il me faudrait, de plus, transférer toute cette quincaillerie d'une machine à l'autre à chaque fois qu'il me prendrait l'envie de changer d'écran ou de processeur ! Et encore, je préfère passer dans un silence miséricordieux sur les problèmes d'accès à l'interface parallèle qui surgissent dès que plus d'un dongle vient s'interposer entre la machine et le périphérique!

Je comprends que les concepteurs de HS5 veuillent protéger le fruit de leur labeur mais la méthode est pernicieuse et le résultat incertain. En supposant qu'un programmeur se procure une copie pirate, s'il tient un tant soit peu à la qualité des logiciels qu'il conçoit, il devra forcément acquérir le produit complet, ne serait-ce que pour disposer du support technique et de la documentation. Protection ou pas, ça ne change rien! Si la copie piratée tombe entre les mains d'un amateur, d'un hacker, pour reprendre le terme consacré, je ne vois pas ce que l'absence de protection changera. Il n'y a pas de manque à gagner, puisque de toute façon cet individu n'aurait jamais acheté HS5 plombé et que la probabilité pour qu'il en fasse un usage professionnel est négligeable.

En fait, le seul effet de la protection est d'indisposer une partie des clients éventuels. En ce qui me concerne, la version protégée est définitivement condamnée, malgré tous ses avantages, à collecter la poussière tout au fond d'une étagère.

PC Soft
216, rue des Escarceliers
BP 3019
34034 Montpellier - Cedex 01
Tél. (16) 67 032 032
Télécopie : (16) 67 03 07 87

TEST	Z386/16	Poly386-33	Poly486-33
Fichier	7h 6"	2h 13"	2h 01"
Landmark	11.2	25.5	80.3
Dhrystones	3793	9103	22758
Whestones	47k	101k	4118k
Vidéo CPS	6885	11652	30296
NORTON SI	18.0	37.5	51.6
NORTON DI	13.1	6.0	6.5
MANIFEST	8.3	20.2	32.0
Chips&Techno	8.97	18.7	26.37
MIPS	2.14	4.47	6.27

elle peut afficher 256 couleurs en 800x600 et 1024x768 caractères, en mode entrelacé ou non, avec le rafraîchissement de 70Hz. Vos yeux émerveillés ne vont pas se fatiguer! Le réceptacle des disques accepte 4 unités pleine hauteur et 2 en 1/2 hauteur, qui peuvent contenir les disques durs, 5"1/4, 3"1/2 ou des dérouleurs de bande, installés à la verticale en position desktop. Le châssis est enfermé dans une carrosserie enfilable, les boutons de redémarrage et autres (sans aucun effet) se trouvent sur la face avant : de quoi donner des sueurs à un technicien confronté à des utilisateurs aux mains baladeuses, qui aiment abuser du bouton « RESET ». Somme tout, Poly386-33PF est un 386 classique qui se distingue par une construction solide et l'équipement sans reproche.

Notre coeur a vraiment battu pour Poly486-33AS. Tout d'abord, une bonne électronique mérite une bonne mécanique. Qui ne connaît pas de problèmes des certains « compatibles » sans marque, en provenance d'un douteux atelier taiwannais, dont la carte mère souffre toutes les contorsions possibles chaque fois qu'on pose un moniteur sur le boîtier ! La machine Polywell est enfermée dans un boîtier de fabrication américaine, de solidité d'une carrosserie d'un tank. C'est une « tour », conçue et fabriquée pour protéger au mieux l'intérieur de toute agression extérieure. Même les connecteurs des câbles extérieurs, situés sur le dessus de la machine, sont enfermés sous une solide plaque coulissante - pas de risque d'arrachement ou d'écrasement accidentel. La face avant est protégée par une porte : des boutons de redémarrage et de vitesse ne sont pas accessibles à n'importe qui, ce qui peut éviter plus d'un ennui à votre réseau local ou multi-poste. Une fois le cache d'en haut enlevé et la porte ouverte, vous pouvez dévisser la plaque de la coté gauche donnant l'accès aux entrailles de la machine. La première chose qu'on aperçoit en bas de la caisse est une énorme alimentation (250 Watts, approuvée UL/CSA) dont les deux puissants ventilateurs ronronnent gentiment pour refroidir la carte mère fixée au-dessus. Les composants (et notamment i486 lui-même) sont refroidis par l'air pulsé en parallèle avec les cartes d'extension, rien ne gêne le flux d'air. La carte mère elle-même, fabriquée en USA pour Polywell par Micronics, donne l'impression d'une excellente qualité - les tests le prouvent. Pas de fils ni rafistolages de dernière minute.

En fait, la vitesse de calcul dépende non seulement de la vitesse du processeur lui-même, mais aussi de la rapidité des mémoires. Le cache intégré de i486, couplé au cache statique extérieur, assurent au processeur les temps de réponse des mémoires à la mesure de ses performances sans d'états d'attente - un i486 à 33 MHz perdrait quelque 10% de sa vitesse avec des mémoires lentes ou mal organisées. Le test de vitesse de la mémoire par MANIFEST prouve que c'est notamment grâce à un excellent temps de réponse des mémoires que Poly 486 doit ses remarquables performances. Ce qui distingue cette machine de son « petit » frère est un BIOS d'American Megatrends Inc version 486 1.2F. D'autres composants de la machine étaient identiques, d'ailleurs Polywell assure pouvoir configurer votre machine en fonction de vos besoins. Le réceptacle des disques apporte 6 emplacements pleine hauteur modulables, qu'on peut remplir à volonté avec des unités 1/1 hauteur, 1/2 hauteur ou des gros disques durs hors dimensions, installés à la horizontale - le choix du pro.

Quelles sont les conclusions ? Si aujourd'hui vous devez choisir une nouvelle machine, n'hésitez pas à

investir dans un i486. Pour quelque milliers de francs de plus vous allez réellement sentir la différence. Et si vous ne voulez pas faire exploser votre budget, essayez un Polywell, un constructeur qui mérite la confiance d'un professionnel.

Polywell Europe
255 bd Robert Ballanger
2 allée du Cerf
Central Parc
93420 Villepinte
téléphone : (1) 49 63 32 88
télécopie : (1) 48 61 18 26

Poly486-33 (pour assembleurs)

Intel486DX à 33 MHz
1 Mo RAM
128Ko cache
1 lecteur disquette
2 série 1 parallèle
clavier Cherry 102 touches
approuvé FCC classe A
manuel utilisateur en français
manuel technique en anglais
19130 F. HT

Poly386-33PF

Intel386DX 33 MHz
4 Mo RAM
64Ko cache
disque dur 80Mo IDE 17ms Maxtor 32Ko mémoire cache
contrôleur IDE
1 lecteur 3"1/2 1.44 Mo
1 lecteur 5"1/4 1.2 Mo
2 série 1 parallèle
VGA ProDesigner II 1Mo
moniteur couleur multisync VGA EIZO
clavier Cherry 102 touches
MS DOS 4.01
manuel utilisateur en français
manuel technique en anglais
22940 F. HT

Poly486-33 (pour assembleurs)

Intel486DX à 33 MHz
1 Mo RAM
128Ko cache
1 lecteur disquette
2 série 1 parallèle
clavier Cherry 102 touches
approuvé FCC classe A
manuel utilisateur en français
manuel technique en anglais

19130 F. HT

NOS TESTS
commentaires

1. Test de Fichier mesure le temps de manipulations extensives d'un fichier de 25 Mo sous FoxPRO 1.0. Plus l'indice étant élevée, moins le système est performant.

2. Landmark version 1.14 indique combien de fois le système est plus rapide qu'un IBM PC à 4.77 MHz, sans accès disque ni opérations mathématiques. Plus l'indice étant élevée, plus le système est performant.

3. Mesure de Dhrystones, Whetstones et vitesse vidéo (en caractères par seconde - CPS) par Diagsoft Inc. ver 1.02. Plus l'indice étant élevée, plus le système est performant.

4. NORTON CI version 4.50 indique combien de fois le système est plus rapide qu'un IBM PC à 4.77 MHz, Plus l'indice étant élevée, plus le système est performant.

5. NORTON DI version 4.50 mesure la vitesse de disques par rapport à un disque 10Mo d'un IBM XT à 4.77 MHz. Plus l'indice étant élevée, moins le système est performant.

6. MANIFEST de Quarterdeck version 1.0 indique la vitesse de la mémoire (premier mégaoctet) par rapport à la mémoire d'un IBM PC à 4.77 MHz. Plus l'indice étant élevée, plus le système est performant.

7. Test de Chips & Technologie ver 1.20 mesure la vitesse des exécutions des instructions et mouvements de données entre le CPU et la mémoire, par rapport à un IBM PC à 4.77 MHz. Plus l'indice étant élevée, plus le système est performant.

8. MIPS par Chips & Technologies ver 1.20 indique le nombre d'instructions générales par seconde. Plus l'indice étant élevée, plus le système est performant.

Tous les tests ont été exécutés sous DR DOS version 5.0. Sur le histogramme (par LUCID 3D), les indices ont étés linéarement réajustées, pour le besoin de la présentation.

I
D
E
E

IDEE DU JOUR : SPELLWIN

" Un correcteur pour Windows "

par G. Korwin

Dans cette nouvelle rubrique, intitulée « IDEE », nous allons publier les idées diverses sur les programmes ou dispositifs à réaliser. Des idées géniales, imprévisibles, nouvelles, créatives, farfelues (qui sait?) en somme, la diversité des idées à réaliser.

En effet, DEBUG MAGAZINE manque (comme tous les autres, par le temps qui court) de pétrole, mais pas d' idées. Chaque programmeur se pose quotidiennement la question : que faire lorsque je terminerai le programme en cours ? Ce programme n'est jamais terminé, mais à des moments rompus, on commence une nouvelle réalisation...

Tous, nous avons aussi des besoins. Tous les jours, nous nous disons : « ah, si cela existait, je serai moins ennuyé »!

Décrivez-nous vos besoins, envoyez les idées, même celles qui semblent le plus bizarre. Qui sait ? Peut-être allez vous déclencher une découverte révolutionnaire ?

Petit à petit le monde d'applications WINDOWS s'enrichit des nouveaux-venus. Le dernier en date est EXCELL 3.0, un superbe nouveau tableur de Microsoft.

EXCELL est l'exemple de ce que doit être une application sous WINDOWS : il n'est pas seulement riche en puissantes fonctions, il est aussi convivial. Vous allez pouvoir l'utiliser intuitivement : les boutons comme la somme automatique ou la mise en forme automatique représentent les fonctions que les utilisateurs ont toujours voulus.

Cependant, EXCELL n'est pas un simple tableur. D'ailleurs, la notion de tableur perd de plus en plus de sa signification originale. En plus d'un tableur, dans EXCELL vous allez trouver des fonctions de gestion de données, un mini-traitement de texte, un outil de dessin, la faculté d'annotation et autres. En fait, EXCELL représente un style non procédural, déclaratif, d'utilisation d'une application. L'utilisateur crée lui-même une application comme il la sent, sans l'analyse préalable. Dans la même catégorie on trouve aussi ObjectVision de Borland, un outil de programmation graphique. D'autres applications semblables vont certainement suivre.

Dans toutes ces applications, l'utilisateur peut créer des textes. Et comme tout texte est sujet à des erreurs, il a besoin d'un correcteur orthographique et grammatical. Ce type de logiciel existe pour des traitements de texte en mode caractère, actuellement il n'existe aucun pour l'environnement WINDOWS.

C'est l'objet de notre idée du mois : un correcteur orthographique et grammatical universel pour WINDOWS. SPELLWIN (tel est son nom proposé) doit fonctionner avec tous les logiciels WINDOWS, qu'ils soient des traitements de texte, tableurs, outils de dessin, bases de données ou d'autre applications, comme ObjectVision. Dans le cas d'EXCELL, par exemple, il doit vérifier les libellées des colonnes, lignes, tableaux, et surtout, des notes, annotations et commentaires attachés.

L'homogénéité de l'environnement WINDOWS permet de créer une telle application universelle.

LE PROCHAIN NUMERO

Debug Magazine n° 14 (juin 1991)

Parution le 7 juin 1991

Sujet : SYSTÈMES MULTI-TÂCHE, MULTI-UTILISATEURS

Cahier Dr Dobb's Journal :
- Un kernel multi-tâche préemptif par Allen Holub
- Un kernel multi-tâche en C++ par Tom Green
- Un make parallèle pour DESQview par Mark Streich
- DDL sous WINDOWS par M Johnson et M Solinski

Et Aussi :
- Capture de données en temps réel. Par Ph Dalléas
- Choisir son réseau. Appel d'offres. Par G Korwin
- Dessin 3D : Coup de Thé. Par Jean Pascal Côte
- Initiation : Récursivité. Par Jean Christophe Riat
- Programmation WINDOWS (suite). Par Serge Grun

Et comme d'habitude :

Podium : Ordinateurs Polywell
Idée : ??? !
Page de John C Dvorak

SONDAGE PERMANENT

Nos lecteurs ont attribué les scores suivants aux articles parus dans DEBUG MAGAZINE n° 11:

1° De Pascal à C++ par Jeff Duntemann

2° Kermit pour OS/2 par BR Anderson (DDJ)

3° Turbo PAscal 6.0 par Serge Grun

Nous félicitons les auteurs!

POUR LA NUMEROTATION DE LA SERIE DEBUG MAGAZINE A DECIDE DE NE PAS UTILISER LE NUMERO 13

DEBUG MAGAZINE vous invite à faire partager votre expérience avec d'autres lecteurs. Nous sommes intéressés par la publication de textes choisis consacrés à la programmation ou à la technologie en général. Nous pouvons publier vos textes qui apportent une information pertinente à d'autres lecteurs. Votre travail sera rémunéré - demandez notre Dossier Collaborateur.

DEBUG MAGAZINE s'intéresse à vous. Nous aimerions recevoir des informations sur les nouvelles technologies, les nouveaux produits ou les développements de votre entreprise. Envoyez-nous vos communiqués de presse, par courrier ou par télécopie. Et si vous voulez donner un coup de projecteur sur une info (ou une vacherie fondée à dire), ne vous gênez pas.

VOUS ETES DYNAMIQUE

Debug Magazine recherche pour son service publicité

ATTACHE (E) COMMERCIAL (E)

* Connaissance de l'industrie informatique souhaitée.
* Expérience des outils bureautiques (traitement de texte, bases de données, tableurs) nécessaire.

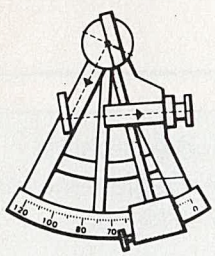
REMUNERATION MOTIVANTE EN FONCTION DES RESULTATS

Pour prendre rendez-vous, envoyez un Curriculum Vitae.

GOTO INFORMATIQUE

Leader des logiciels de communication PC.

- *Vous êtes surchargé d'idées, et vous ne supportez pas les structures contraignantes ...*
- *Vous pensez que la micro-informatique est un superbe univers d'expression créative ...*
- *Nous nous ressemblons : sans cesse à la recherche de nouveaux concepts, peut être un peu fous, en tout cas propulsés quotidiennement par nos passions !*



Vous avez l'enthousiasme en poupe ! Alors prenez le cap GOTO.

Pour répondre à notre croissance et renforcer notre équipe, nous recherchons des

DEVELOPPEURS MICROS

Autonome (ça ne veut pas dire individualiste), vous serez responsable de la création ou de l'évolution de logiciels.
Curieux, doué d'esprit d'initiative et rigoureux, vous êtes capable de mener à bien vos projets de développement micro dans leur intégralité. Bien sûr vous avez déjà développé avec succès, mais vous souhaitez aujourd'hui vivre l'aboutissement des projets qui vous sont confiés.
Passionné de micro-informatique, de formation Ingénieur (Bac+5), vous êtes en début de carrière ou confirmé et avez une bonne connaissance du C et MS-DOS.

Si cette opportunité correspond à votre ambition, merci d'adresser votre dossier de candidature (lettre + CV + photo) à l'attention de Francis Piet.

GOTO Informatique - Château de la Bonnerie - 111, rue de Croix B.P.55 - 59510 HEM

PETITES ANNONCES LECTEURS

Je vends ordinateur **AMSTRAD PC 2286/40** (286/12 Mhz, disque dur 40Mo), écran couleur 14 HRCD (piqué 0,29), lecteurs 5"25 et 3"5, nombreux logiciels : 5 000 F. Michel Barraud - bureau : 42 92 46 36, domicile : 45 47 28 28.

Vous avez des besoins en informatique (formation, conseil, développement d'applications...). Alors, téléphonez au : (1) 30 38 05 04.

Commerçants, artisans, professions libérales... Vous voulez informatiser votre entreprise, alors n'hésitez pas à téléphoner au : (1) 30 38 15 04.

Vends **Carte Fax-Modem** pour PC 9600 BAUDS (Modem 2400 BD) : 2 500F. Tél : 94 63 74 52.

Vends **Turbo Pascal PRO. 5.5** (Turbo Pascal, Débogueur, Assembleur) en 6 volumes et 8 disquettes : 900F. Carte mère 386/25 avec 4 Mo 70 ns : 5 800F. Tél : 49 23 30 88 (Châtellerault).

Vends **Disque dur SEAGATE**, format 5" 1/4 pleine hauteur, capacité non form : 383Mo, temps d'accès : 14,5 mns + **carte Contrôleur ESDI - WD 1007A** + câbles : **le tout n'a jamais servi.** Prix : 7 500F à débattre.

Vends disquettes 3" 1/2 HD 1,44 Mo
Prix : 100F la boîte de 10.
Tél : (1) 39 46 53 26. (Ludovic)

Vous pouvez commander **Help III version Shareware** pour 20F TTC, port compris, à Cumps. BP 51, 40501 Saint Sever Cedex. Précisez 5"1/4 ou 3" 1/2 (didacticiels MS-Dos).

A vendre **Source PASCAL GAG Virus** parlant : 30F. Adresser chèque A2M1 - 58, rue R. Ballon - 33700 MERIGNAC. Association d'aide aux personnes handicapées par l'utilisation de la micro-informatique. Gonzalez Michel.

Vends calculatrice HP15C neuve + manuel, interface FAG (liaison RS232C, imprimante, magnétophone à cassettes). Prix à débattre. Jérôme Delcourt - 62, rue Negrier - 59800 LILLE. Tél : 20 31 83 63.

Pour les abonnés de DEBUG Magazine, les petites annonces sont gratuites (joindre la bande témoin du dernier numéro reçu). Pour les autres lecteurs, 50 f, par coupon renvoyé. Règlement par chèque ou carte bancaire à l'ordre de: DEBUG Magazine - 3, rue de l'Atlas - 75019 Paris.

J'ACHETE

J'ECHANGE

JE VENDS

DEBUG MAGAZINE

BULLETIN DE PETITES ANNONCES

☐ J'ÉCHANGE ☐ J'ACHÈTE ☐ JE VENDS

☐ Je suis abonné à DEBUG MAGAZINE : ci-joint bande témoin du dernier numéro

☐ Je ne suis pas abonné à DEBUG MAGAZINE; ci-joint un chèque de 50

pliez ici en second et collez l'autre côté au ruban adhésif.

AFFRANCHIR
AU TARIF
LETTRE..

DEBUG MAGAZINE
3, rue de l'Atlas
75019 Paris

pliez ici en premier.

QUESTIONNAIRE – LECTEUR

Année de naissance : 19__
Sexe : ☐ M ☐ F

Votre secteur d'activité :

- ☐ Agriculture
- ☐ Enseignement
- ☐ Autres Fonction Publique, Collectivité locale
- ☐ Association
- ☐ Energie
- ☐ Métallurgie
- ☐ Chimie, verre, plastique
- ☐ Construction automobile, mécanique, navale, aéronautique
- ☐ Electronique
- ☐ Bâtiment, travaux publics
- ☐ Autres secteurs industriels
- ☐ Commerce
- ☐ Transports
- ☐ Télécommunications
- ☐ SSII
- ☐ Autres services aux entreprises
- ☐ Hôtellerie, réparation et autres services aux particuliers
- ☐ Banque, Assurance
- ☐ Autres secteurs commerciaux

Nombre de salariés dans l'établissement : ____

Votre profession :

- ☐ Propriétaire
- ☐ PDG-DG-Gérant
- ☐ Cadre
- ☐ Employé
- ☐ Ouvrier
- ☐ Retraité
- ☐ Autre

Votre matériel :

- ☐ IBM ou compatible
- ☐ Apple, MAC
- ☐ Mini
- ☐ Mainframe

L'utilisation de votre matériel :

- ☐ privée
- ☐ professionnelle
- ☐ les deux

Etes-vous satisfait de votre équipement actuel ? ☐ Oui ☐ Non
Allez-vous l'étendre ? ☐ Oui ☐ Non

Vos commentaires :

BULLETIN D'ABONNEMENT *2 ANS COMPLET*

❒ Je m'abonne pour **22** numéros + **22** disquettes listing + **2** numéros spéciaux : **990 F** au lieu de **2 002 F**
Je règle par : ❒ chèque à l'ordre de **Debug Magazine**
❒ carte bancaire
Nom de la carte : .....
N° ..../..../..../..../..../..../..../..../..../..../..../..../..../..../..../ Date d'expiration /..../..../..../..../
Nom :..... Prénom : .....
Société :.....
Rue :.....
Lieu :.....
Code postal :..... Ville : .....
Pays :..... Tél : .....
Date :..... Signature : .....
L'abonnement commencera au n° .....

A renvoyer à **DEBUG MAGAZINE** - 3, rue de l'Atlas - 75019 Paris

-50 %

BULLETIN D'ABONNEMENT *2 ANS SIMPLE*

❒ Je m'abonne pour **22** numéros + **2** numéros spéciaux : **530 F** au lieu de **968 F**
Je règle par : ❒ chèque à l'ordre de **Debug Magazine**
❒ carte bancaire
Nom de la carte : .....
N° ..../..../..../..../..../..../..../..../..../..../..../..../..../..../..../ Date d'expiration /..../..../..../..../
Nom :..... Prénom : .....
Société :.....
Rue :.....
Lieu :.....
Code postal :..... Ville : .....
Pays :..... Tél : .....
Date :..... Signature : .....
L'abonnement commencera au n° .....

A renvoyer à **DEBUG MAGAZINE** - 3, rue de l'Atlas - 75019 Paris

-45 %

BULLETIN D'ABONNEMENT *1 AN SIMPLE*

❒ Je m'abonne pour **11** numéros + **2** numéros spéciaux : **290 F** au lieu de **484 F**
Je règle par : ❒ chèque à l'ordre de **Debug Magazine**
❒ carte bancaire
Nom de la carte : .....
N° ..../..../..../..../..../..../..../..../..../..../..../..../..../..../..../ Date d'expiration /..../..../..../..../
Nom :..... Prénom : .....
Société :.....
Rue :.....
Lieu :.....
Code postal :..... Ville : .....
Pays :..... Tél : .....
Date :..... Signature : .....
L'abonnement commencera au n° .....

Anciens numéros à **35 F** pièce : ❒ n° 2 ❒ n° 3 ❒ n° 4 ❒ n° 5 ❒ n° 6 ❒ n° 7 ❒ n° 8 ❒ n° 9
Anciens numéros à **37 F** pièce : ❒ n° 10 ❒ n° 11
Hors Série n° 1 : 58 F.

A renvoyer à **DEBUG MAGAZINE** - 3, rue de l'Atlas - 75019 Paris

-40 %

DEBUG Magazine

MARCHE PC

LA PUBLICITE A DES PRIX ABORDABLES POUR TOUS !!!

Vous avez des produits ou services qui méritent d'être connus.
Vous voulez les promouvoir par des campagnes de publicité (achat ou vente).
Vous cherchez des rubriques spécifiques, claires, pratiques et faciles à trouver par vos prospects .
Vous désirez optimiser au maximum votre investissement.
Vous manifestez l'enthousiasme de réaliser des opérations suivies et répétées dans l'année.
Vous voulez un marché unique en France avec une transparence des prix incontestable.
Mais, vous ne disposez pas de budgets faramineux.

VOTRE SOLUTION : LES RUBRIQUES « MARCHE PC » !!!

1/2 PAGE : 3.950 F. HT

Marché PC de DEBUG Magazine, l'emplacement de choix pour trouver les produits référencés.

Renseignez-vous au 42.02.29.16

Demandez le Service Commercial

1/16 PAGE : 490 F. HT

TARIFS1991

1 PAGE : 7.900 F. HT
1/2 PAGE : 3.950 F. HT
1/4 PAGE : 1.980 F. HT
1/8 PAGE : 990 F. HT
1/16 PAGE : 490 F. HT

CONDITIONS DE VENTE

1. Couleurs du support + 20 %
2. Frais techniques en sus : forfait 500 F. HT
3. Paiement à la commande.
4. Insertion possible jusqu'à 10 jours avant parution.

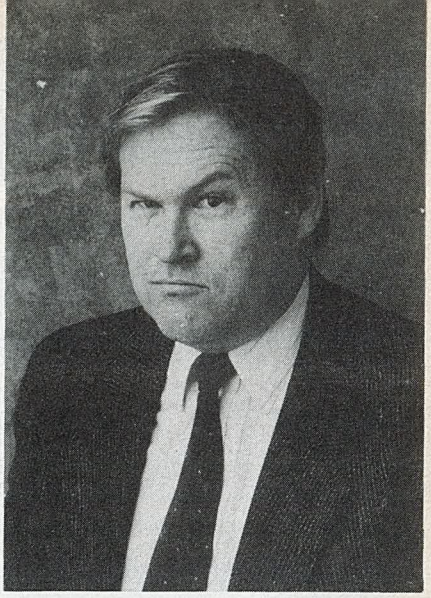
1/8 PAGE : 990 F. HT

DEGRESSIFS

5 Parutions par an : 5 %
10 Parutions par an : 10 %
15 Parutions par an : 15 %
20 Parutions par an : 20 %

1/4PAGE : 1.980 F. HT

John C. Dvorak



QUATRE CLEFS POUR UN SERVICE SIMULE

Tom Petersen, auteur, conférencier et critique, aime rouspéter publiquement au sujet des entreprises modernes qui ont perdu le sens de trois éléments fondamentaux des affaires : client, qualité de produits et, particulièrement, le service. Son truc favori est d'aller dans les locaux d'une entreprise pour fouiller les paniers à papier et retirer les notes de service. Ensuite, il peut s' étendre sur le sujet, décrivant comment aucune de ces correspondances ne mentionne jamais le client, la qualité de produit ni « le service ».

Les critiques cyniques ou les sociologues découvrent toute cette matière et lèvent le doigt accusateur sur nos chers leaders de l'industrie et des affaires. Je dois approuver, car j'ai contribué à la description de certains parmi ces caractères.

Prenons, par exemple, IBM (s'il vous plat!). C'est une entreprise parmi celles qui réussissent le mieux dans le monde d'affaires. Elle est connue plus pour ses services que pour tout autre de ses activités, excepté, peut-être, la vente. Les faits sont là : le service vend et IBM a du succès à cause du service.

Par contre, dans le milieu de petits ordinateurs, il n'y a pas beaucoup à dire sur le service. La dernière tendance à la mode est de le faire payer bien qu'il devrait être gratuit. C'est le sort de l'attitude suivante : « Voici notre produit, mais si vous voulez qu'il marche, vous n'avez qu'à payer davantage ! » Aux USA, il y a une grand débat au sujet de l'assistance technique vendue grâce à des numéros de téléphone payants, commençant par le 900. Quand vous appelez ces numéros 900, votre compte téléphonique est débité au prorata du temps. Encore une fois, le service est cher et le logiciel lui-même, bon marché.

Alors, quelle est la solution ? Chaque fois que je pense cette question, je me rappelle ce qu'en dit l'acteur et comique, George Burns. Selon un client habitué d'un amusant restaurant à L.A. proche de chez moi, assis à table, Burns expliquait les secrets pour être un bon acteur. Il disait : « La clef pour être un bon acteur est simple - sincérité et honnêteté ». Après cela, il faisait une pause et continuait : « Si vous pouvez simuler cela - vous avez gagné ! ».

Alors, peut-être qu'un service illusoire peut être aussi efficace que le service réel. Après tout, par le temps qui court, l'image est tout. Plus je pense à ce sujet-là, plus je suis convaincu que de plus en plus d'entreprises utilisent déjà cette technique du « service simulé ». Voici quelque traits caractéristiques que j'ai observé:

1) La technique appelée « Bon Marché ». Rappelez-vous que la parole, indéniablement, ne coûte rien. Une entreprise de logiciel qui a du succès va propager partout le message qui insiste sur la « qualité de service» qu'elle fournit. Le motto de Président doit être : « Le service est pour nous la priorité n° 1 ». C'est une méthode peu chère pour impressionner le public, sans effectuer quelque travail réel que ce soit.

2) La technique : « Nous sommes débordés parce que nous avons du succès ». N'ayez plus qu'une ligne téléphonique pour chaque dix millions de chiffre d'affaires réalisé. Les lignes de l'assistance technique doivent obligatoirement passer par un vieux commutateur mécanique, branché sur un répondeur qui va répéter : « Toutes nos lignes sont occupées. Veuillez rester en ligne, un technicien va vous répondre ». Cela donne à votre client une impression que vous êtes occupé (ce qui techniquement veut dire que vous avez du succès). Le système doit être programmé pour créer automatiquement une attente de l'ordre de dix minutes. Heureusement, la plupart de clients va abandonner avant que ce délai expire.

Évidemment, il y a toujours l'éventualité que cette impossibilité à parler à quiconque va à la fin provoquer un mécontentement publique. Habituellement, quelque journaliste déplaisant va exploiter le sujet dans un article acerbe. C'est a ce moment que le Président va intervenir en public et affirmer : « Nous sommes très attachés à la solution de ce problème. Nous avons commandé les nouvelles lignes téléphoniques et nous avons engagé des nouveaux techniciens. Le problème a surgi car nous sommes parmi les rares entreprises qui offrent un service illimité et gratuit aux développeurs qui utilisent nos produits ». Il est bien entendu qu'après cela, aucun changement ne doit être effectué. Si un autre journaliste arrive avec les mêmes critiques, on recommence la même manoeuvre et on gagne encore davantage de la bonne publicité.

3) La technique :« Faites-les rappeler ». Supposez qu'un fanatique reste suspendu sur la ligne jusqu'a qu'un employé de service répond enfin. Que devez-vous faire alors? D'abord, tous les employés chargés de répondre doivent être des fraîches émigrés Serbo-croates de l'Hercegovine Basse, parlant le français avec un fort accent américain, de Chicago (un plus certain lors de recrutement, si vous tombez sur une telle rareté). Peu importe la question posée, l'employé doit répondre : « Ça, c'est problème technique. Vous attendez. Je transfère appel ». Ensuite, il doit raccrocher. Les gens présument toujours que les systèmes téléphoniques ratent la moitié des transferts alors le fait de raccrocher va être interprété comme une erreur du système.

4) La technique : « La Pèche au Futé ». Il y aura toujours un futé plus tenace que les autres, qui va trouver la méthode pour contourner le système et arriver finalement à quelqu'un qui doit répondre à la question. Alors, ayez toujours quelqu'un qui va prendre le message et promettre d'appeler en retour. Trouvez alors ou celui qui appelle se trouve et rappelez-le durant les heures de déjeuner, tôt le matin ou tard le soir, laissant le message avec un numéro « spécial » « réservé » qui est relie sur le dit serbo-croate.Reconnaissez-vous une de ces techniques?

[SERVICE LECTEURS : INTERESSANT = 42 / MOYEN = 43 / MAUVAIS = 44]]

3614 DEBUG
LANCEMENT LE 15 MAI 1991
QUITTEZ LE MONDE DE LA COMPLEXITÉ

E SERVICE TECHNIQUE POUR LES PROFESSIONNELS

614 DEBUG est un nouveau service télématique, spécialement crée pour vous.
uisez dans les ressources de DEBUG MAGAZINE, immédiatement et facilement.

E LIEU DE RENCONTRE DES EXPERTS

614 DEBUG est l'endroit de rencontre d' experts, comme vous. La messagerie permet de discuter, lancer des forums, échanger des idées.
vous ne trouvez pas de réponse ni dans les livres ni dans les manuels, posez la question à d'autres : la réponse peut arriver instantanément!

OMMUNIQUEZ AVEC LA RÉDACTION

614 DEBUG est un moyen facile pour dialoguer avec la rédaction. Posez des questions, suggérez des idées, postez vos lettres, commentaires et pinions.

OTRE ESPACE PUBLICITAIRE GRATUITE

ue vous ayez un lecteur à échanger contre une carte ou que vous cherchiez une pièce détachée rare, un équipement d'occasion à prix réduit, protez des petites annonces **3614 DEBUG.** Un partagiciel à annoncer, un logiciel à proposer en vente, vous cherchez un éditeur, programmeur, onsultant ou dépanneur - profitez de leur présence sur **3614 DEBUG,** annoncez vos offres et vos besoins. Vos petites annonces seront lues instannément, sans attendre la sortie mensuelle du magazine. **3614 DEBUG,** c'est instantané et ça peut rapporter gros!

ELECHARGEZ

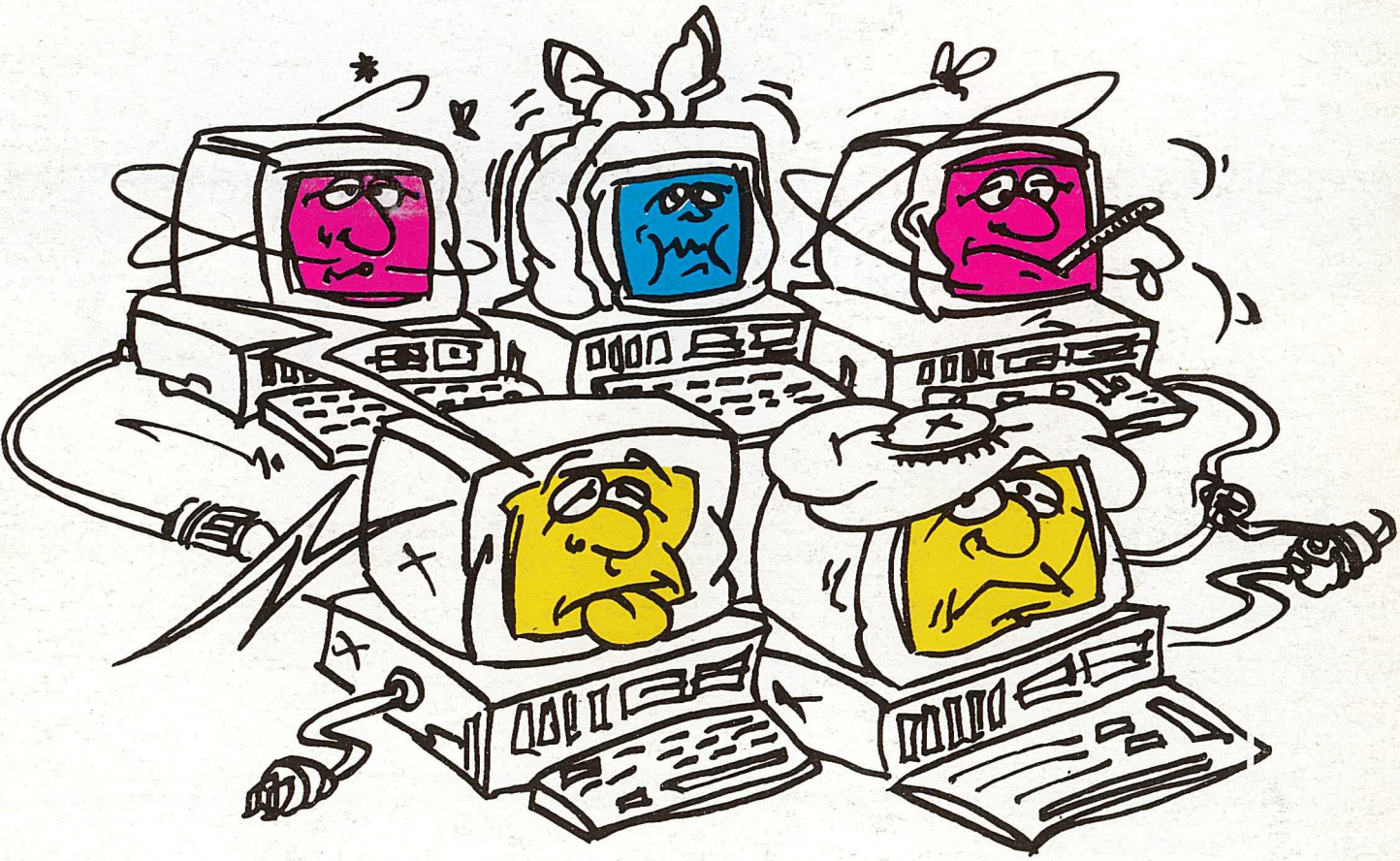
614 DEBUG est un moyen rapide pour télécharger les listings de DEBUG MAGAZINE. Vous pouvez télécharger tous les sources et exécutables vec le logiciel OVERCOM (émulation Minitel, TTY, VT52/VT100, protocoles GOTO, Kermit, X et Y modem, SM1, langage de procédure, autoprentissage. Fonctionne sur toutes les cartes modem et avec un Minitel, livré avec câble Minitel, adaptateur DB9/DB25, manuel 300 pages). ontactez-nous : prix spécial pour les lecteurs de DEBUG MAGAZINE!

ENDANT LE LANCEMENT LE SERVICE EST GRATUIT!

endant deux mois de démarrage, vous pouvez utiliser ce service gratuitement - seul le coût de communication sera à votre charge (tarif 3614).

BUGS? DEBUG!

DEBUG Magazine offre chaque mois les solutions techniques aux professionnels. Ils lisent DEBUG Magazine. Et vous?



abonnez-vous!